System-Level Design Methodologies for Networked Multiprocessor Systems-on-Chip

Virk, Kashif Munir

Publication date:
2008

Document Version
Publisher's PDF, also known as Version of record

System-Level Design Methodologies for Networked Multiprocessor Systems-on-Chip

Kashif Virk

Kongens Lyngby 2008
IMM-PHD-2008-193

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

Summary

Brevity is the soul of wit
William Shakespeare

The first part of the thesis presents an overview of the existing theories and practices of modeling and simulation of multiprocessor systems-on-chip. The systematic categorization of the plethora of existing programming models at various levels of abstraction is the main contribution here which is the first such attempt in the published literature.

The second part of the thesis deals with the issues related to the development of system-level design methodologies for networked multiprocessor systems-on-chip at various levels of design abstraction with special focus on the modeling and design of wireless integrated sensor networks which are an emerging class of networked embedded computer systems.

The work described here demonstrates how to model multiprocessor systems-on-chip at the system level by abstracting away most of the lower-level details albeit retaining the parameters most relevant at the system-level. The multiprocessor modeling framework is then extended to include models of networked multiprocessor systems-on-chip which is then employed to model wireless sensor networks both at the sensor node level as well as the wireless network level.

In the third and the final part, the thesis covers the issues related to the design, implementation and testing of a system-on-chip-based wireless sensor node development platform, specifically, for the Hogthrob project. This part also deals with the cycle-accurate model of the multiprocessor system-on-chip and its pos-

sible extensions to the transaction-level model.

The thesis, as a whole makes contributions by describing a design methodology for networked multiprocessor embedded systems at three layers of abstraction from system-level through transaction-level to the cycle accurate level as well as demonstrating it practically by implementing a wireless sensor node design.

Resumé

Denne afhandling indledes med en præsentation af eksisterende teoretiske og praktiske metoder til modellering og simulering af multiprocessor system-on-chip designs. Det primære formål er - for første gang i litteraturen - at danne et samlet overblik over de mange programmeringsmodeller, der findes på forskellige abstraktionsniveauer.

I det følgende afsnit behandles problemstillinger omkring udvikling af system-level design metodikker for netværksbaserede multiprocessor system-on-chip designs. Der fokusers især på modellering og design af trådløse integrerede sensor-baserede netværk, som finder større og større anvendelse i embeddede computer systemer.

Dette arbejde demonstrerer, hvorledes et multiprocessor system-on-chip design kan modelleres på systemniveau ved at ignorere de detaljer og parametre, der har mindre afgørende betydning for den overordnede funktion. Dette simple framework kan derefter udvides ved at inkludere modeller af netværksbaserede system-on-chip designs. På dette grundlag kan der uadarbejdes modeller for generelle wireless sensor netværk, både på sensor node niveau og på det trådløse netværks niveau.

I afhandlingens tredie og sidste del beskrives først design, derpå implementering og endelig test af en system-on-chip baseret trådløst sensor udviklingsplatform beregnet for Hogthrob projektet. Afslutningsvist omtales en cycle-accurate model af multiprocessor system-on-chip design og de tilhørende udvidelsesmuligheder for en tilsvarende model på transaction-level.

Afhandlingen beskriver metoder til design af netværksbaserede embeddede mul-

tiprocessor systemer på tre abstraktionsniveauer: System-, transaction-, og cycle-accurate niveau, og demonstrerer endvidere en praktisk implementering af et trådløst sensor node design.

Preface

This thesis was prepared at the Department of Informatics & Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark in partial fulfillment of the requirements for acquiring the Ph.D. degree in engineering. The Ph.D. project was supervised by Professor Jan Madsen and funded by the Danish National Research Council (STVF) project titled "Hogthrob - Networked System-on-a-Chip Nodes for Sow Monitoring" (STVF 2059-03-0027).

The thesis consists of a summary report and eleven chapters including a collection of eight research papers authored during the period of the Ph.D. project, and elsewhere published.

Lyngby, December 2007

Kashif Virk

List of Publications

The following publications have been based on the research work carried out during the Ph.D. project and are, therefore, included in this Ph.D. thesis:

A: Kashif Virk and Jan Madsen. Computing Platforms - Multiprocessor Modeling and Simulation. The ARTIST Roadmaps for Research and Development. (Editors: Bruno Bouyssounouse and Joseph Sifakis). *Lecture Notes in Computer Science, Volume 3436, 2005.* Section 29, Pages: 388-406. Springer Scientific Publishers. Published.

B: Jan Madsen, Kashif Virk and Mercury Gonzalez. Abstract RTOS Modeling for Multiprocessor System-on-Chip. *Proceedings of the IEEE International Symposium on System-on-Chip* (SoC'03), November 2003. Pages: 147-150. Published.

C: Jan Madsen, Shankar Mahadevan, Kashif Virk and Mercury Gonzalez. Network-on-Chip Modeling for System-Level Multiprocessor Simulation. *Proceedings of the IEEE Real-Time Systems Symposium* (RTSS'03), December 2003. Pages: 265-274. Published.

D: Kashif Virk and Jan Madsen. A System-Level Multiprocessor System-on-Chip Modeling Framework. *Proceedings of the IEEE International Symposium on System-on-Chip* (SoC'04), November 2004. Pages: 81-84. Published.

E: Kashif Virk, Knud Hansen and Jan Madsen. System-Level Modeling of Wireless Integrated Sensor Networks. *Proceedings of the IEEE International Symposium on System-on-Chip* (SoC'05), November 2005. Pages: 179-182. Published.

F: Kashif Virk, Mohammad Shafique, Jan Madsen and Aric Menon. System-Level Modeling and Simulation of MEMS-based Sensors. *Proceedings of the IEEE International Multi-Topic Conference* (INMIC'05), December 2005. Pages: 1-6. Published.

G: Kashif Virk, Martin Leopold, Andreas Vad Lorentzen, Martin Hansen, Phillipe Bonnet and Jan Madsen. Design of a Development Platform for HW/SW Codesign of Wireless Integrated Sensor Nodes. *Proceedings of the IEEE EuroMicro Conference on Digital System Design* (DSD'05), August 2005. Pages: 254-260. Published.

H: Kashif Virk and Jan Madsen. Functional Testing of Wireless Sensor Node Designs. *Proceedings of the IEEE International Conference on Innovations in Information Technology* (Innovations'07), November 2007. Pages: 123-127. Published.

The following research publications are not included in this Ph.D. thesis but, nevertheless, form the basis for the research work carried out during the Ph.D. project.

I: Kashif Virk and Jan Madsen. Resource Allocation Model for Modeling Abstract RTOS on Multiprocessor Systems-on-Chip. *Proceedings of the IEEE NorChip Conference* (NorChip'03), November, 2003. Pages: 48-51. Published.

J: Jan Madsen, Kashif Virk and Mercury Gonzalez. A SystemC-based Abstract Real-Time Operating System Model for Multiprocessor System-on-Chip. *Multiprocessor Systems-on-Chips, 2004.* (Editors: Wayne Wolf and Ahmed Jerraya). Chapter 10, Pages: 283-311. Morgan-Kaufmann Publishers. Published.

K: Jan Madsen, Shankar Mahadevan and Kashif Virk. Network-Centric, System-Level Model for Multiprocessor System-on-Chip Modelling. *Interconnect-Centric Design for Advanced SoC and NoC, 2004.* (Editors: Jari Nurmi, Hannu Tenhunen, Jouni Isoaho and Axel Jantsch). Chapter 13, Pages: 341-365. Kluwer Academic Publishers. Published.

L: Shankar Mahadevan, Michael Storgaard, Jan Madsen and Kashif Virk. ARTS: A System-Level Framework for Modeling MPSoC Components and Analysis of their Causality. *Proceedings of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS'05), September 2005. Pages: 480-483. Published.

M: Shankar Mahadevan, Kashif Virk and Jan Madsen. ARTS: A SystemC-based Framework for Multiprocessor System-on-Chip Modeling. *Design Automation for Embedded Systems, 2007.* Volume 11, Number 4, Pages: 285-311. Springer Scientific Publishers. Published.

Acknowledgements

Chance favors the prepared mind
Louis Pasteur, Lille (France) 1854

I thank my Ph.D. project supervisor, Professor Jan Madsen for giving me the chance and guiding me throughout my Ph.D. project. I would also like to thank Mercury Gonzalez who initially developed the MPSoC Modeling Framework as a part of his Master's thesis project and which has subsequently been extended by myself and Michael Storgård and Knud Hansen under Jan Madsen's active supervision. I am also grateful to Marcus Schmitz for sharing his data on the *task graphs* which have been extensively used by me and many of my fellow researchers at IMM for simulations. Thanks to Shankar Mahadevan for sharing his insights on Networks-on-Chip and for numerous meetings and countless e-mails regarding our joint publications on the MPSoC Modeling Framework. The participants of the course, *Design & Synthesis of Embedded Systems*, given during 2004 and 2005, deserve special mention with whom I enjoyed many brilliant discussions on Wireless Sensor Networks while I was their teaching assistant during my Ph.D. Thanks to Sardjan Capkun and Professor Mani Srivastava for their time and efforts to arrange my visit to UCLA as a research visitor though I never ended up being there.

Finally, my thanks go to my wife and daughter for tolerating my odd working hours and bearing the sight of our living room constantly littered with books and papers.

Kashif Virk
Bagsværd, December 2007

Contents

Chapter 1

Introduction

The most profound technologies are those that disappear.
Mark Weiser, Xerox, PARC (USA) 1991

1.1 System-Level Modeling of Networked Embedded Computer Systems

As more embedded computer systems are being integrated into system-on-chip (SoC) designs and as the interactions of concurrent (and, possibly, real-time) software with embedded parallel and distributed computing platforms becomes more complex, the embedded computer systems designers must reason about:

- computing platform design for programmability,
- co-execution of hardware-like and software-like system-level behaviors,
- system-level performance impacts of hardware architectures that execute the software functionality.

Many of the critical system-level design decisions are those that involve the anticipation of hardware/software interactions; as hardware is loaded with the

software functionality, software is deployed onto a variety of hardware resources (architectures) and parts of a system (mixed hardware and software) must interact with the yet-to-be-designed rest of the system (which may also include mixed hardware and software).

Modeling systems in the large is an important trend in systems engineering and it plays a central role there. The purpose of modeling is to build models of systems which satisfy given requirements. The use of models can profitably replace experimentation on actual systems with incomparable advantages such as:

- enhanced modifiability of the model and its parameters;
- ease of construction by integration of models of heterogeneous components;
- generality by using genericity, abstraction, and behavioral non-determinism,
- enhanced observability and controllability, especially, avoidance of the probe effect and of disturbances due to experimentation;
- possibility of analysis and predictability by the application of formal methods.

Building system-level models which faithfully represent complex systems is a non-trivial problem and a pre-requisite to the application of formal analysis techniques. Usually, modeling techniques are applied at the early phases of system development and at a higher level of abstraction. The need for a unified view of the various lifecycle activities of an embedded computer system and of their interdependencies have motivated the so-called model-based system design approaches which rely heavily on the use of modeling methods and tools to provide support and guidance for system development and validation.

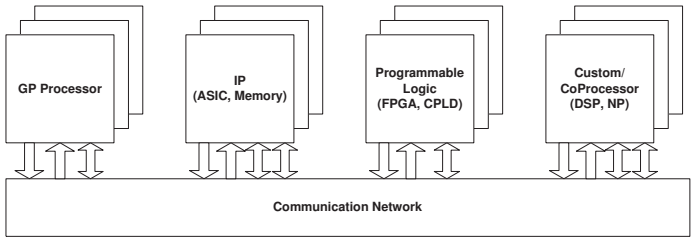


Figure 1.1: *A typical Multiprocessor System-on-Chip architecture*

Most of the future embedded computer systems are likely to be designed for real-time applications that execute on multiple processors (Figure 1.1). Modern embedded computer systems also exhibit an increasingly large quantity of communication capabilities. The communication infrastructures in a multiprocessor

embedded computer system may comprise either local, internal communication infrastructures (e.g., in a multiprocessor computer system, the processing elements can be connected through shared memory, dedicated communication links or a communication network) or global, external communication infrastructures (e.g., an interface to an external network which can either have wired or wireless links) or both.

This gives rise to two possible categories of such systems:

- **Embedded Network Systems**
  This category of embedded computer systems are characterized by the fact that they have a communication network embedded in them for handling internal communications. The codesign of these systems requires modeling of not only the hardware and software parts of the embedded computer system but also the embedded communication network which, essentially, involves modeling of the network geometry[1]. Networks on Chip are an emerging example of this class of parallel multiprocessor computer systems-on-chip.

- **Networked Embedded Systems**
  In this category of embedded computer systems, each computing platform forms a node that has access to a communication network for handling communications external to the computing platform. So the network interface becomes a part of the node design. The nodes and the network are together embedded in the environment. The design of such a distributed multiprocessor computer system is not only focused on a particular network device but also on consideration of the interactions between the nodes because it is very important to be able to predict, measure and verify the real-time attributes of the entire distributed multiprocessor computer system. When the prediction or extrapolation of such real-time attributes is impossible (due to the complexity of the system, its non deterministic nature, etc.) the availability of a proper model can make the difference. Therefore, in addition to *hardware/software* partitioning step described above for embedded network systems, the codesign of such systems demands another *hierarchical* partitioning step which involves:

  1. node-level modeling
  2. network-level modeling

  In addition, a model of the environment with which the networked embedded system interacts, can serve to verify and, possibly, validate the involved algorithms and architectures. The modeling domain dichotomy

[1] network geometry comprises network topology (network hardware) as well as network protocols (network software).

arising between the inherently continuous environment models and the essentially discrete networked embedded system models, can, possibly, be alleviated through hybrid systems modeling.

Wireless Sensor Networks constitute a typical example of this class of distributed multiprocessor computer systems-on-chip.

Orthogonal to the above categorization, an alternate classification of embedded computer systems can be as:

- *multifunction systems* - that can concurrently operate across multiple application domains, e.g., mobile multimedia terminals can capture video data, process audio streams, browse the web simultaneously, or
- *multimode systems* - that can operate in several alternative modes of operation, e.g., mobile phones can accommodate several communication protocols [4], or
- *hybrid systems* - that are both multifunction as well as multimode.

For the sake of completion, it is worth mentioning that, as a flexible variation of the embedded computer systems categorized above, **adaptive embedded computer systems** tend to achieve optimum computation and/or communication load distribution either through an internal reconfigurable network (to meet varying computation load requirements - as in reconfigurable computer systems) or through an external reconfigurable network (to meet varying communication requirements - as in mobile ad hoc networks) or both.

In modeling parallel and distributed multiprocessor computer systems, the operating systems have a major role. A parallel multiprocessor system is *tightly coupled* so that the global status and workload information on all processors can be kept current at a low cost. The system may use a centralized scheduler. When each processor has its own scheduler, the decisions and actions of the schedulers of all the processors are coherent. In contrast, a distributed multiprocessor system is *loosely coupled*. In such a system, it is costly to keep the global status and workload information current. The schedulers on different processors may make scheduling and resource access control decisions independently. As a consequence, their decisions may be incoherent as a whole.

This thesis report attempts to describe the work carried out during the Ph.D. project that attempts to show how to model and evaluate parallel and distributed multiprocessor computer systems in their completeness at various levels of abstraction as well as at different levels of hierarchy.

SystemC has been selected as the modeling language to model hardware, software as well as network geometry. Since SystemC is based on C++, it is possible

to formulate an executable specification of the modeled system which is, essentially, a program that behaves in the same way as the SystemC specification of the system. This avoids inconsistencies and errors and helps to ensure completeness of the specification.

1.1.1 Modeling Multiprocessor Systems on Chip

The embedded computer system designers, usually, use processor-based templates to build today's system-on-chip (SoC) designs, which contain one or more processor cores with considerable on-chip memory and sophisticated communication infrastructures. Because on-chip processor cores are often either legacy or third-party components, the designers need correct functional models to accurately track the interaction of processor core(s) with the rest of the embedded system.

The embedded hardware designers use Hardware Description Language (HDL) simulators to validate their work, but these simulators model the processor micro-architecture in too much detail to efficiently simulate complex processor cores. The embedded software designers, on the other hand, routinely use cross-development toolkits containing a cross-compiler and an instruction-set simulator (ISS) to validate functionality and assess application performance. Thus, exploring and validating a complex SoC design requires a single, integrated hardware-software cosimulation platform. The academic research groups, as well as the electronic design automation vendors, have developed numerous such platforms.

Traditional cosimulation design environments use multi-language system descriptions - HDL for hardware and C (or similar languages) for software - to construct an efficient link between event-driven hardware simulators to cycle-based ISS's.

Therefore, there has been a need for a system design language that describes the functionality of both hardware and software. It must allow the system to be defined, first without making assumptions about the implementation, and then to be refined into the exact implementation with hardware and software components. It is also important to be able to use standard models of computation (MOCs) at the initial design stages. Further, one may not wish to concretely specify the communication mechanisms and instead leave it to be defined by the underlying operational semantics of the MOCs being deployed.

More recently, using C/C++ for hardware design descriptions and design flows has gained popularity because using the same language for describing hardware

and software can, potentially, bridge the gap between hardware and software description languages. Using the same language also makes it possible to simulate the entire system within a single simulation engine.

SystemC is the leader in system-level modeling with C++. The SystemC approach consists of a progressive refinement of specifications. SystemC allows both applications and platforms to be expressed at sufficiently high levels of abstraction while, at the same time, enabling the linkage to hardware implementation and verification. SystemC has the potential to provide a full-fledged description of an execution platform which can serve as the target of a codesign methodology. Thus, SystemC is a viable intermediate representation language.

SystemC describes the functionality of both hardware and software inside a unified specification language based on C++. At a high level of abstraction, SystemC allows the use of a common language for software and hardware specifications and simulation of the whole system. However, one of the problems encountered with SystemC 2.0 is the lack of features to support embedded software modeling. For some classes of applications modeled with SystemC, it is not, currently, possible to completely model the software behavior of the targeted architecture.

The availability of RTOS models is becoming strategic inside HW/SW (hardware/software) co-design environments. Apart from providing some assurances about the timely performance of tasks, an RTOS provides a very useful abstraction interface between applications with hard real-time requirements and the target system architecture. Indeed, for the simulation of software modules, such as preemption and/or priority-based scheduling, generally present in any RTOS, the SystemC simulator does not offer all the necessary functionalities. This is because, during simulation, the RTOS scheduler, responsible for determining which thread will run next, manages both software and hardware threads identically. It means that systems with hard real-time constraints requiring an RTOS (Real-Time Operating System) based on a preemptive priority-based kernel cannot be modeled in a natural manner. As a consequence, a joint refinement of the software and hardware parts is a tedious task in SystemC 2.0.

To support the designers of single chip-based embedded systems (which includes multiprocessor platforms running dedicated RTOS's) to easily simulate various hardware/software configurations, at high-level, as a part of this Ph.D. project, we have successfully developed an abstract RTOS modelling environment based on SystemC by abstracting the real-time operating system features at the system level. In our abstract RTOS modeling framework, we deal with generalized abstract tasks and processing elements. Our abstract RTOS system model deals with the analysis of the execution behaviour of real-time applications running on a heterogeneous multiprocessor computing platform. In our model, such an

application is represented as a multi-threaded application comprising a set of abstract tasks with certain essential execution parameters. Each task can either execute independently or precede a given set of other tasks. Moreover, each task also excludes a given set of other tasks for the use of shared resources.

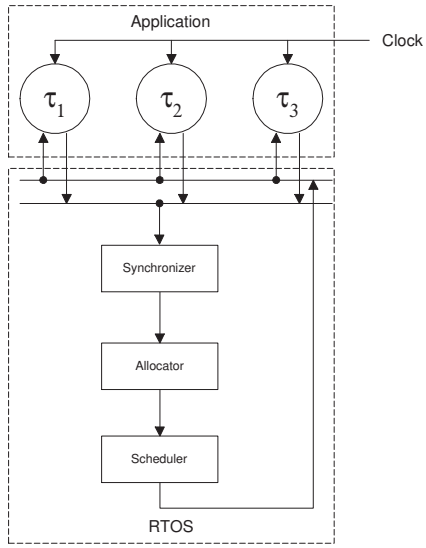


Figure 1.2: *Abstract RTOS Model*

Based on the principle of composition, three distinct but closely-related RTOS services have been modeled, namely, task scheduling, execution synchronization, and resource allocation (Figure 1.2). The scheduler is modelled around the priority-based preemptive scheduling policy which is one of the most preferred scheduling policies for the execution of tasks in real-time systems due to its higher schedulability. In our scheduler model, which supports, RM scheduling, EDF scheduling or other variants, whenever a task becomes ready or finishes execution, the scheduler is invoked and it then looks for a ready task with maximal priority to continue execution. In our synchronizer model, synchronization is regarded as a means to prevent undesirable task interleavings by the scheduler. Our synchronizer model is responsible for establishing the correctness of the results computed by the multiprocessor platform and it implements the Direct Synchronization (DS) protocol. Unfortunately, most mechanisms used in the basic RTOS services are not compositional in nature. Even if a mechanism can provide assurances individually to each task, there is no systematic way to provide assurances for an aggregate of two except in trivial cases. One manifestation of this problem is *priority inversion*. To partly offset this problem, the resource allocator model is based on the priority inheritance protocol.

The results have shown that simulation overhead introduced by the RTOS model is negligible while providing modeling accuracy.

1.1.2 Modeling Networks on Chip

With the growing complexity of embedded systems and the capacity of modern silicon technology, there is a trend towards heterogeneous architectures consisting of several programmable and dedicated processors, implemented on a single chip, known as a System-on-Chip (SoC). As an increasing portion of applications is implemented in software which, in turn, is growing larger and more complex, dedicated operating systems will have to be introduced as an interface layer between the application software and the hardware platform. On the other hand, the hardware platform will either be developed as a part of the design process or configured from an existing reconfigurable platform, which allows for the implementation of parts of an application as dedicated processors (ASIC's).

Modern silicon technologies, with minimum device geometries in the nanometer range (<100nm), have made it possible to integrate hundreds of processors on a single chip. In these deep submicron technologies, the on-chip interconnection fabric is a major source of delay and power consumption which is challenging the on-chip communication infrastructure and forcing a change from device-centric to interconnect-centric design methodologies. Traditionally, on-chip communication has either been conducted via dedicated point-to-point links or by shared media like a bus. Neither is very suitable for generalized communication handling in large systems. A promising solution is to have a dedicated, segmented, and, possibly, packet-switched network fabric on the chip, a Network-on-Chip (NoC) [1].

Hence, when mapping an application onto its target platform, hardware/software codesign aspects [5] have to be taken into account. These include mapping of tasks onto software, hardware, or a combination of both, as well as task dependencies on the communication infrastructure. In order to do so, accurate modeling of the systems and all the interrelationships among the diverse processors, software processes and physical interfaces and interconnections, is needed. One of the the primary goals of system-level modeling is to formulate a model within which a broad class of designs can be developed and explored. To support the designers of single-chip based embedded systems, which includes multiprocessor platforms running dedicated real-time operating systems (RTOS's) as well as the effects of on-chip interconnect network, a system-level modeling/simulation environment is required to support an analysis of the:

- consequences of different mappings of tasks to processors (software or

hardware),

- network performance under different traffic and load conditions,
- effects of different RTOS selections, including various scheduling, synchronization and resource allocation policies.

As a part of this Ph.D. project, we have developed a NoC modeling environment based on SystemC which can provide the SoC designers a software-like, system-level abstraction of the computing platform as well as supporting the three requirements mentioned above for system-level design-space exploration.

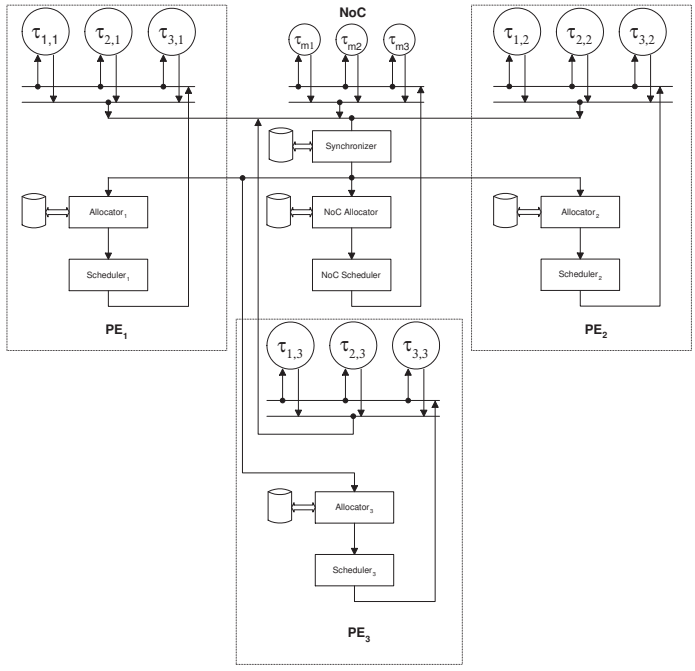


Figure 1.3: *NoC Model*

The multiprocessor SoC model developed earlier has been extended to handle the effects of on-chip interconnection infrastructure, i.e., network-on-chip (Figure 1.3). We model a generic multi-threaded application, running on a multiprocessor computing platform under the control of one or more abstract RTOS's and extend the model with a model of an on-chip communication network which can provide provisions for run-time inspection and observation of the on-chip communication. Instead of dealing with each specific application and system architecture, we deal with generalized abstract tasks, processing elements, and communication infrastructures. This not only broadens the applicability of our

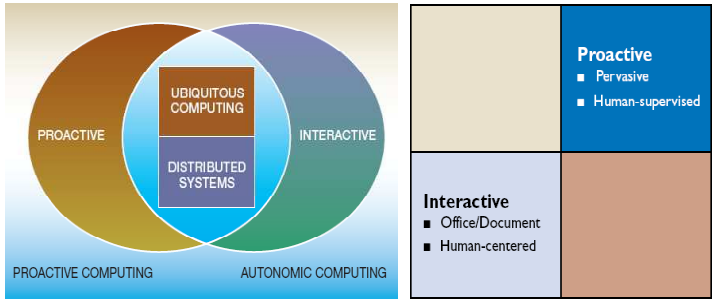
modeling framework, but also leads to a better understanding of the problem at hand. Using this system-level design approach, implementations of the most promising network alternatives can be prototyped and characterized in terms of performance and overhead. Taking communication into account during hardware/software mapping is essential in order to obtain optimized solutions.

We have also demonstrated the capabilities of our modeling framework by modeling and simulating an example of a multifunction embedded network system which has a hand-held multimedia terminal application mapped on a heterogeneous 4-processor SoC architecture interconnected through a torus on-chip network topology. It is worth mentioning, however, that our system-level modeling framework supports more sophisticated scheduling policies and NoC topologies. Moreover, features like including the effects of the network interface and memory accesses as well as dynamic load balancing support can be built upon by adding more components to the existing framework components.

1.1.3 Modeling Wireless Sensor Networks

Over the 50 years of modern computing, a new class of computers has emerged about once a decade, progressing through mainframe computers, mini computers, personal computers, and mobile hand-held computers. Each successive computing paradigm has relied upon technological advances, especially levels of integration governed by Moore's law, to make computing available in a form factor not previously possible. Each has ushered in new uses for computer technology. Each succeeding generation is smaller, more plentiful and more intimately associated with personal activity than the generation that preceded it. However, the new trend in modern computing is not only how to keep pace with Moore's law but also how to deal with the consequences of its decades-long reign.

With each passing year, a given computing capacity becomes exponentially smaller and cheaper because the prolonged exponential growth in the semiconductor process technology has enabled the number of transistors on a cost-effective semiconductor chip and, therefore, the processing or storage capacity of that chip, doubles every year or two, following Moore's law. While it has provided ever more computing power, this technology is now being applied in ways that enable a new computing paradigm - *proactive computing* (Figure 1.4). The proactive computer systems are a class of **networked embedded computer systems** which are pervasively coupled with the the environments in which they are embedded using sensors and actuators to both monitor and shape their physical surroundings.



(a) (b)

Figure 1.4: *A comparison of computing paradigms [8, 3].*

The semiconductor manufacturing techniques that underlie this miniaturization can also be exploited to build micro-mechanical structures that can sense forces and fields in the physical world as well as exceptionally small radio transceivers. These inexpensive, low-power sensing, computing and communication devices can be deployed throughout a physical space, enabling sensing, processing and wirelessly communicating this information. Combining these hardware capabilities with the system software technology that forms the Internet makes it possible to narrow the gap between the physical and the virtual spaces with increasing fidelity.

The density of instrumentation made possible by a shift to mass-produced intelligent sensors and the use of pervasive networking technology gives these wireless sensor networks a new kind of scope that can be applied to a wide range of uses. These applications can be roughly categorized into:

- space monitoring,
- object monitoring, and
- monitoring the interactions of objects with each other and the encompassing space.

The first application category includes environmental and habitat monitoring, precision agriculture, indoor climate control, surveillance, treaty verification, and intelligent alarms. The second includes structural monitoring, ecophysiology, condition-based equipment maintenance, medical diagnostics, and urban terrain mapping. The most dramatic applications involve monitoring complex interactions, including wildlife habitats, livestock behavior, disaster management, emergency response, ubiquitous computing environments, asset tracking, healthcare, and manufacturing process flow.

However, bridging the gap between the hardware technology's raw potential and the broad range of applications presents a systems design challenge[2]. The individual devices in a wireless sensor network (WSN) are inherently resource constrained: they have limited processing speed, storage capacity, and communication bandwidth. These devices have substantial processing capability in the aggregate, but not individually, so their many vantage points on the physical phenomena must be combined within the network itself such that the aggregate performs sophisticated functions. The network must allocate limited hardware to multiple concurrent activities, such as sampling sensors, processing, and streaming data.

Because they are so closely coupled to a changing physical space, the sensor nodes forming the network will experience wide variations in connectivity and will be subject to potentially harsh environmental conditions. Their dense deployment, generally, means that there will be a high degree of interaction between the sensor nodes, both positive and negative. The potential interconnections between devices must be discovered and information routed effectively from where it is produced to where it is consumed. Each of these factors further complicates the design of wireless networking protocols.

There must also be a means of programming the ensemble. Because manually configuring large networks of small devices is impractical, the sensor nodes must organize themselves and provide a means of programming and managing the network as an ensemble, rather than administering individual devices. Despite these operational factors, deploying and maintaining the sensor nodes must remain inexpensive.

To realize the opportunity offered by this new computing paradigm, the information technology must address a new collection of challenges. The wireless sensor networks merge a wide range of information technology that spans hardware, systems software, networking, and programming methodologies (Figure 1.5).

A wireless sensor node's hardware consists of sensors, analog-to-digital converters (ADCs), a microprocessor, data storage, a data transceiver, device controllers that tie the pieces together, and an energy source. Recently, a new operating point has emerged that suits all these components. As semiconductor circuits become smaller, they consume less power for a given clock frequency and fit in a smaller area. In simple microcontrollers, process scaling increases efficiency rather than adding functionality, allowing them to operate near one milli-watt while running at about ten MHz. Most of the circuits can be powered off, so the standby power can be about one micro-watt. If such a device is active one percent of the time, its average power consumption is just a few micro-watts.

However, low-power microprocessors have limited storage, typically, less than

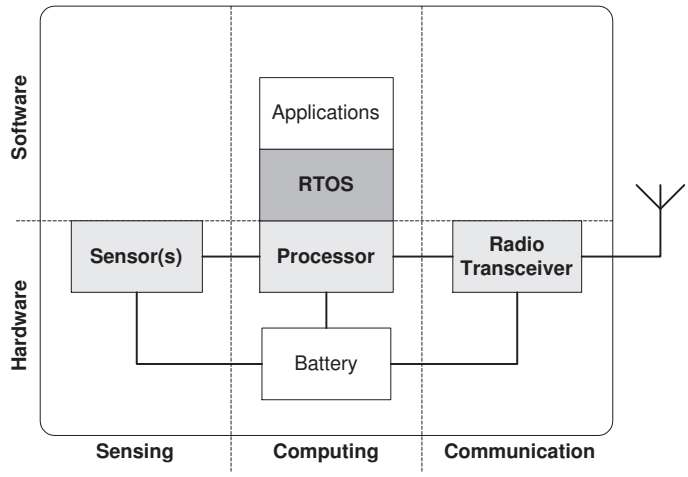


Figure 1.5: *Wireless Sensor Node*

10 Kbytes of RAM for data and less than 100 Kbytes of ROM for program storage - or about 10,000 times less storage capacity than a portable computer has. This limited amount of memory consumes most of the chip area and much of the power budget. Typically, larger amounts of flash storage is incorporated, perhaps, a megabyte, on a separate chip.

Sensors and actuators have undergone a revolution with the emergence of microelectromechanical systems (MEMS) technology. Micro-Electro-Mechanical Systems (MEMS) can sense a wide variety of physical phenomena cheaply and efficiently. The processes for etching transistors on silicon can be used to carve out tiny mechanical structures, such as a microscopic springboard within an open cavity. Gravitational forces or acceleration can deflect this cantilevered mass, causing powerful internal forces that cause changes in material properties or delicate alignments, which can be amplified and digitized. The sensed signal is, typically, in the form of a voltage signal which is converted by an ADC into a binary number that a microcontroller can store or process. The first major commercial MEMS sensor, the accelerometer, has been used by automotive manufacturers to trigger automotive airbag release. Whereas high-precision piezoelectric accelerometers cost hundreds of dollars, MEMS-based sensors provided sufficient precision for a few dollars. Once the devices entered mass production, they could ride the CMOS technology growth of modern chips to become increasingly accurate while remaining inexpensive. A wide variety of MEMS devices can sense various forces, chemical concentrations, and environmental factors.Many more sophisticated structures have been developed to detect other phenomena. These structures consume a few milli-watts and only need to be

turned on a fraction of the time. Extremely efficient ADCs have been developed so that the sensor subsystem has an energy profile similar to the processor.

WSN radio transceivers consume about 20 milli-watts and their range, typically, is measured in tens of meters. For small devices to cover long distances, the network must route the information hop by hop through nodes, much as routers move information across the Internet. Even so, communication remains one of the most energy-consuming operations, with each bit costing as much energy as about 1,000 instructions. Thus, WSNs process data within the network wherever possible. In addition to that, to minimize energy consumption, just like most of the device's components, the radio transceiver will likely be turned off most of the time.

Nevertheless, the scale of power typically consumed by all the subsystems of the device described above can be obtained in many ways. Batteries remain the primary energy storage devices and there have been substantial improvements in battery technology with improved storage density, form factor, and recharging. A typical cubic-centimeter battery stores about 1,000 milliamp-hours, so centimeter-scale devices can run almost indefinitely in many environments. Although the energy storage technology has advanced substantially it has not improved at the pace associated with silicon-based processing, storage, and sensing. However, the emergence of alternative storage devices, such as ultracapacitors and miniaturized fuel cells with high energy density is promising. Moreover, energy harvesting mechanisms are being actively developed. Solar cells generate about 10 milli-watts per square centimeter outdoors and 10 to 100 micro-watts per square centimeter indoors. Mechanical sources of energy, such as the vibration of windows and air conditioning ducts, can generate about 100 micro-watts. In most deployment settings, the network must operate for long periods of time and, as the sensor nodes are wireless, so the available energy resources - whether batteries, energy harvesting, or both - limit their overall operation.

Energy constraints dominate algorithm and system design trade-offs for small devices. Therefore, to make the networked embedded node an effective vehicle for developing algorithms and applications, a modular, structured runtime environment should provide the scheduling, device interface, networking, and resource management primitives on which the programming environments rest. It must support several concurrent flows of data from sensors to the network to controllers. Moreover, microsensor devices and low-power networks operate bit by bit (or in a few cases, byte by byte), so software must do much of the low-level processing of these flows and events.

During the growth in capability and complexity of these devices, several distinct operating systems approaches have emerged to make application design more manageable. The traditional approach to controller design has been to

hand-code scheduling loops to service the collection of concurrent flow events, but this yields brittle, single-use firmware that has poor adaptability. A more general-purpose solution is to provide fine-grain multithreading. This approach has been extensively researched for general-purpose computation and it can be effectively extended to the tiny, networked sensor regime, because the execution threads that must be interleaved are simple. These requirements have led to a component-based tiny operating system environment which provides a framework for dealing with extensive concurrency and fine-grain power management while providing substantial modularity for robustness and application-specific optimization. The TinyOS[7] framework establishes the rules for constructing reusable components that can support extensive concurrency on limited processing resources.

The data compression and communication scheduling techniques can also conserve energy at lower protocol layers. Some protocol overhead is associated with data communication to maintain routing structures, manage contention, and enhance reliability. The wireless sensor networks can avoid explicit protocol messages by piggybacking control information on data messages and by overhearing packets destined for other nodes. They can use pre-scheduled time to reduce contention and the time the radio transceiver remains live. This can be coordinated with the high-level application behavior by, for example, periodic low-rate data sampling. Alternatively, the network could implement energy conservation, generically, within lower protocol layers by, for example, time division multiple access.

In the spatial dimension, the network can assign specific responsibilities to certain sensor nodes, such as re-transmission or aggregation. Finally, the network can reject uninteresting packets by turning off the radio transceiver after receiving only a portion. However, because these many optimizations can be mutually conflicting, a rich and growing body of research literature employs different combinations of techniques under different application and platform assumptions.

In order to efficiently utilize the extremely limted resources of wireless sensor nodes, accurate modeling of the key aspects of wireless sensor networks is necessary so that system-level design decisions can be made. the design of the sensor nodes requires a deep understanding of their various constituent components, their underlying technologies and the interactions between those components. The wireless sensor network design space consists of the choice of different *software components* - application code and real-time operating system - *hardware components* - processor, memory, radio transceiver, A/D converter, sensors, battery and application-specific devices - and *network parameters* - network topology, network protocols, number of nodes, their role, etc. Such components may be either common-off-the-shelf or subject to the design process. To al-

low modeling of the whole sensor node architecture, these components should be representable through models. The modeling languages should be specific for the different components to ease their representation, re-use, synthesis, and validation. Furthermore, it is desirable to have a uniform modeling language to provide joint cross-layer optimization of the different parts of the system required by the challenging constraints of wireless sensor networks described above.

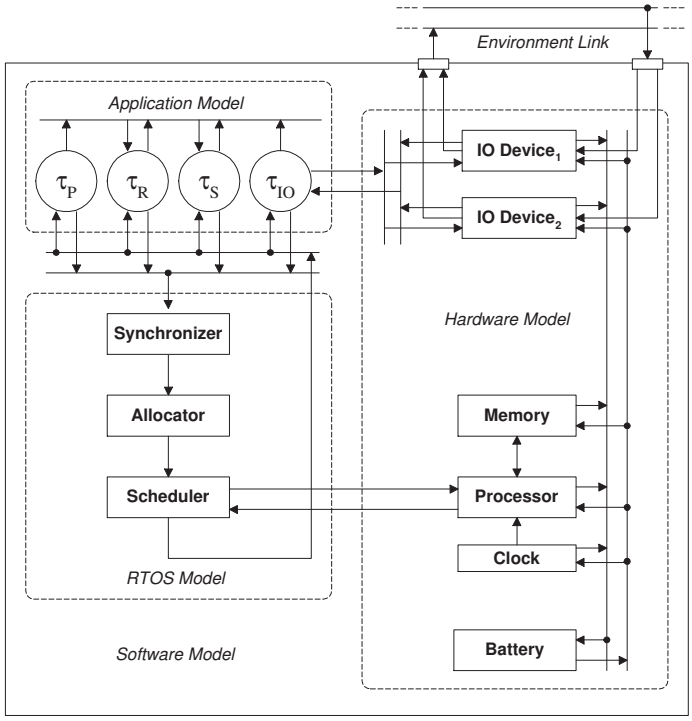


Figure 1.6: *Wireless Sensor Node Model*

To provide the wireless sensor network designers a system-level abstraction of the sensor network for system-level design-space exploration, we have extended our earlier work on SystemC-based abstract multiprocessor SoC modeling framework and developed a modeling framework that enables system-level modeling of sensor network behavior by modeling the applications, real-time operating system, sensors, processor, and radio transceiver at the sensor node level and environmental phenomena, including radio signal propagation, at the sensor network level. The concepts of SystemC hierarchical channels have been employed to develop a methodology for modeling the radio and sensor channels that can accurately model the wireless sensor network-related phenomena like radio ir-

regularity and radio interference. In order to make a seamless transition from a system-level sensor node model to an implementation-level (cycle-accurate) sensor node model, the concepts for a bridging model have been developed that makes use of Transaction-Level Modeling (TLM) at Level 1 (TL1) for modeling the various serial bus protocols common on wireless sensor node platform designs. The bridging model refines the HW/SW partitioning by modeling the processor using an instruction set simulator (ISS) which interacts with the RTOS services model. Finally, cycle-accurate implementation-level models have been developed using HW/SW codesign for the processor and its interfaces in VHDL, the MEMS-based accelerometer in VHDL-AMS and the custom hardware block for Kalman Filtering in Matlab/VHDL. The system-level modeling framework is more generic while the bridging model and the cycle-accurate models are specific to the Hogthrob Project described in the following sub-section.

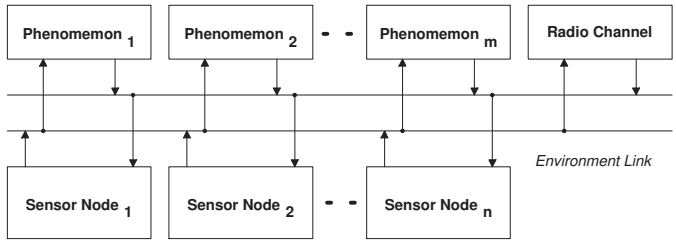


Figure 1.7: *Wireless Sensor Network Model*

WSNs appear to represent a new class of computing. They follow the trends of size, number, and cost, but have a markedly different function. Rather than being devoted to personal productivity tasks, WSNs make it possible to perceive what takes place in the physical space in ways not previously possible. In addition to offering the potential to advance many scientific research pursuits, they also provide a vehicle for enhancing larger forms of productivity, such as manufacturing, agriculture, construction, and transportation.

As the technology that is commercially available today becomes established enough to warrant greater investment, straightforward engineering efforts will yield complete devices with sensing, processing, storage and communication functions that fit in much less than a cubic centimeter of space and cost just a few euros.

Looking forward, the technology will likely evolve into a much less distinct and visible form. Instead of being housed in many small devices, these elements will likely become part of the manufacturing process for various materials and objects. These sensors will tend to operate within the ambient energy sources of their intended environment and be placed at key junctures where analysis

is most critical. As this vision evolves, so will the need for fundamentally new information technology architectures, from signal-processing algorithms to programming languages.

1.1.4 Hogthrob Project - A Practical Approach to Wireless Sensor Network Modeling & Design

Apart from myriads of applications being proposed for wireless sensor networks, their low cost and size, ease of deployment, and autonomous operation make them a viable and non-intrusive solution for livestock monitoring applications. Extensive automated methods for detecting oestrus[2] and health disorders have been developed within many livestock production systems. In dairy cows, traits like milk yield, body temperature, walking activity, etc. have been used for the detection of oestrus and health disorders. For group housed sows, automated methods for oestrus detection are based on sows' activity measurements using infrared sensors or accelerometers.

Today's Danish farms for pig production are using RFID tags for sow identification and controlling their food consumption. However, these tags have proven to be quite impractical to locate sows in large pens. Moreover, they are not flexible enough to be useful in contexts other than controlling the food consumption. For example, the pig farmers have to manually monitor the key aspects of a sow's lifecycle such as the onset of oestrus or farrowing - the phenomena that have a profound effect on pig production.

In this context, the Hogthrob Project[6] aims at developing a cheap, robust and energy-efficient wireless sensor network technology adapted to the requirement of sow monitoring. The goal is to develop wireless sensor nodes that can be tagged onto the sows (in replacement of the RFID tags they wear today), a wireless sensor network infrastructure and the software application allowing farmers to track changes in the activity of loose group housed sows prior to oestrus to mate sows at an optimal time. Such wireless sensor nodes should combine sensing, processing and communication abilities on a chip, must be low-cost (costing no more than a couple of Euros), small-sized (small enough, when packaged, to be worn as an ear tag) and low-energy (a few months' autonomy is a minimum).

The project started with the design and development of a wireless sensor node prototype to be used in field experiments (Figure 1.8). In parallel, a model

[2]oestrus or *heat period* is the period when a livestock animal can be bred and it lasts for a short time only. If an animal is not bred during its first oestrus, it is considered unproductive from the commercial point of view since it normally returns to oestrus about 3 weeks later and needs to be fed and housed meanwhile.

of the wireless sensor network has been developed to explore the design space. Input from the field experiments on the wireless sensor node and the wireless sensor network model should lead to a progressive refinement of the wireless sensor network design till it is feasible for a system-on-chip production.

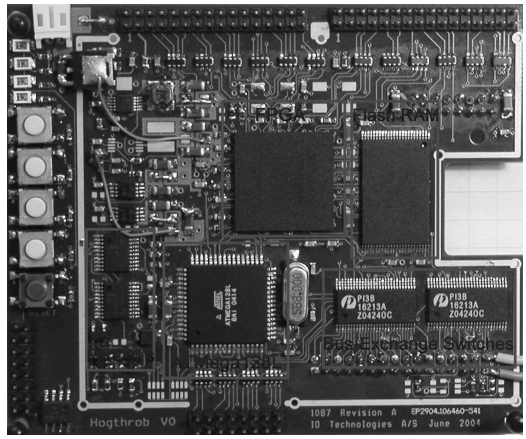


Figure 1.8: *Motherboard of the Hogthrob Sensor Node Hardware*

This project is a consortium between the following partners:

- **LIFE/KU**
  The Department of Large Animal Sciences of the Faculty of Life Sciences (LIFE) at the University of Copenhagen (KU) focuses on monitoring and modeling the behavior of group housed sows for the purpose of detecting behavioral deviations caused by oestrus. (development of automated methods for oestrus detection)
- **IMM/DTU**
  The Department of Informatics and Mathematical Modeling (IMM) at the Technical University of Denmark (DTU) focuses on the modeling and design of the wireless sensor nodes and on the high-level modeling of the wireless sensor network infrastructure.
- **DIKU/KU**
  The Department of Computing Sciences (DIKU) at the University of Copenhagen (KU) focuses on the development of the application software and its interaction with the wireless sensor node hardware.
- **I/O Technologies A/S**
  The I/O Technologies A/S is responsible for the prototyping of the wireless sensor node hardware in collaboration with the IMM/DTU and DIKU/KU.

- **Danish Pig Production Council**
  The Danish Pig Production Council is the customer of the technology to be developed and it partly finances the field experiments and assists LIFE/KU in data acquisition.

The acceleration data acquired during the field experiments constitute the basis for the development of a method for classifying the activity of sows as well as for the development of the oestrus detection application model.

Numerous sensor network research projects have designed sensor nodes with various microprocessors (from Atmel, Hitachi, Intel, etc). However, none of the sensor node architectures, reported so far in the literature, approach the sensor node design from a hardware/software codesign perspective. During this Ph.D. project we have developed a system-level model for modeling wireless sensor networks as well as designed a sensor node development platform in order to explore the design-space both in terms of hardware and software.

1.2 An Outline of the Thesis

This Ph.D. thesis comprises twelve chapters including the introduction (Chapter 1) and the conclusions (Chapter 12). Of the remaining ten chapters, eight chapters (Chapters 2-6 and Chapters 8-10) consist solely of research publications resulting from the work carried out during the Ph.D. project with each research publication forming a chapter. Each research publication is fairly self-contained to justify such an organization for the thesis. Two chapters (Chapter 7 and Chapter 11) have been written exclusively for this Ph.D. thesis and their contents have not been already published elsewhere. These chapters are intended to fill the continuity gaps between the eight research publication-based chapters.

The following sections give an overview of the work carried out through the duration of this Ph.D. project which is described here and the contributions made to the field of research.

1.2.1 System-Level Modeling - Theories and Practices

One of the the primary goals of system-level modeling is to formulate a model within which a broad class of designs can be developed and explored. The first part of this thesis comprises **Chapter 2** which presents an overview of the existing theories and practices of modeling and simulation of multiprocessor

systems-on-chip. A systematic categorization of the plethora of existing programming models at various levels of abstraction is the main contribution here which is the first such attempt in the published literature.

1.2.2 System-Level Modeling of Wireless Sensor Networks

The second part of this thesis comprises Chapter 3 to Chapter 6 and deals with the issues related to the development of system-level design methodologies for networked multiprocessor systems-on-chip at various levels of design abstraction with special focus on the modeling and design of wireless integrated sensor networks which are an emerging class of networked embedded computer systems.

Owing to their small form-factors, ad-hoc deployment and the requirements of extended periods of unattended operation, wireless sensor networks form an extremely resource- and energy-constrained, sensing, computation and communication environment which makes the design and optimization of these systems a complex task. In particular, the design of wireless sensor nodes requires a deep understanding of their diverse constituent components, their underlying technologies and the interactionsbetween those components.

Therefore, to support the designers of sensor networks and, in particular, sensor nodes, a system-level modeling/simulation environment is required to support an analysis of the consequences of different mappings of application tasks to processors (software or hardware); effects of different communication- and routing protocols, and the effects of different RTOS selections, including various scheduling, synchronization and resource allocation policies.

In order to be able to explore the design space at very early stages in the design process, it is important to have an accurate system-level model of the sensor network capturing all the inter-relationships among the diverse processors, software processes and radio- and sensor interfaces.

Chapter 3 describes a SystemC-based abstract RTOS (Real-Time Operating System) modeling framework for system-on-chip platform modeling. The abstract RTOS modeling framework has been developed at the system level by abstracting away most of the lower-level details of a real-time operating system albeit retaining the parameters most relevant at the system-level.

Chapter 5 describes an extension of the SystemC-based abstract RTOS modeling framework for system-on-chip platform modeling to multiprocessor system-on-chip platforms and demonstrates the capabilities of the multiprocessor modeling framework by mapping the task graphs of a multimedia application to the

abstract models of various processing elements and simulating the whole system to check if the tasks meet their local and/or end-to-end deadlines under precedence and resource constraints. This work has also formed the basis for other system-level modeling-related research activities such as networks-on-chip and reconfigurable computing platforms.

Chapter 6 describes a further extension of our earlier work on SystemC-based multiprocessor system-on-chip modeling framework which can provide the wireless sensor network designers a system-level abstraction of the sensor network for system-level design-space exploration to meet the requirements mentioned above.

Though our aim has been to develop a general sensor network modeling environment, we have actually been driven by a real-life sensor network application – the Hogthrob project which, as described above, is concerned with the development of a wireless sensor network infrastructure for sow monitoring.

In our SystemC-based modeling framework, a sensor network model is designed following the principle of composition. We model a sensor network at two levels: the sensor network level and the sensor node level.

At the sensor node level, a sensor node platform model is split into two sections: the software section - for functional simultion of the sensor node platform and the hardware section - to enable estimation of the energy consumption of the sensor node platform.

The software section of the sensor node platform model consists of the application model, comprising a set of task models and the RTOS model, composed of a set of RTOS services.

At the sensor network level, a sensor node platform model is embedded in an environment model that models the environmental phenomena to be sensed by the sensor network application.

To bridge the abstraction gap between the system-level abstract sensor network model (mentioned above) and the implementation-level, cycle-accurate sensor node model (mentioned later), we have introduced an intermediate-level bridging model based on transaction-level modeling concepts that attempts to connect the top-level and the bottom-level models in a consistent manner. This model is described in **Chapter 6**.

1.2.3 Wireless Sensor Node Design & Test and Cycle-Accurate Modeling

The third and final part of this thesis covers the issues related to the design, implementation and testing of a system-on-chip-based wireless sensor node development platform, specifically, for the Hogthrob project. This part also describes the various design-space exploration approaches followed through hardware-software codesign on a cycle-accurate model of the wireless sensor node. However, the details of the various components of this cycle-accurate model are split accross the chapters comprising this part.

Chapter 8 describes the design of the wireless sensor node development platform for the Hogthrob project. It also describes some details of the cycle-accurate model of the wireless sensor node, especially, the microprocessor model. The architectural design space of the wireless sensor node development platform is explored from a hardware/software codesign perspective to end up with a complete wireless sensor node implemented on a single chip.

Chapter 9 describes the testing of the wireless sensor node development platform for the Hogthrob project. Designing wireless sensor nodes for wireless sensor networks is an error-prone and, hence, an iterative process because of the inherent intricacies of designing a wireless communication-oriented, mixed-signal, distributed embedded system. Therefore, it is imperative to follow a systematic design methodology coupled with an efficient test approach to satisfy all the design requirements for the target application.

We have developed a hierarchical, at-speed, functional test methodology and applied it successfully to test the custom-built Hogthrob wireless sensor node development platform. This test methodology, though unique in its approach, extends earlier work in this area and can be applied, in general, for testing all types of wireless sensor nodes. A significant contribution of our work is a unified test methodology for wireless sensor nodes that combines, as well as, extends various component-level and board-level test techniques and exploits the on-board programmable logic for implementing a Test Controller that has a strategic access to all the board-level and component-level interfaces. This chapter also describes the cycle-accurate models of the Flash memory programmer and the various communication interfaces on the Hogthrob platform.

Chapter 10 describes a cycle-accurate model of an acceleration sensor using a mixed-signal extension of VHDL - VHDL-AMS. As a further contribution to the Hogthrob project, we have proposed a model-driven MEMS-based micro-sensor design methodology which is more than a combination of the existing top-down and bottom-up design approaches as it enables MEMS-based micro-

sensor design, validation, and optimization in a consistent, step-by-step manner and is compatible with the existing embedded system design methodologies. We have illustrated the capabilities of our proposed MEMS design methodology by applying it to design, simulate, and optimize a microaccelerometer. The work carried out in this regard gives a good insight into the system-level modeling of microsystems which can stimulate ideas about hybrid systems modeling and verification, especially, in the context of the emerging new area of wireless integrated sensor networks.

Chapter 11 gives the details of a cycle-accurate model of a coprocessor for Kalman filtering. Kalman Filter is the standard DSP tool for combining the information from many sensors as well as low-pass filtering, amplification, etc. A properly-designed Kalman Filter allows one to observe only a few quantities, or measured outputs, and then reconstruct or estimate the full internal state of a system.

In the Hogthrob project, accelerometer data from the field experiments on sows were analyzed for acceleration patterns and an automatic classification method based on a Multi-Process Kalman Filter was implemented by the KVL research group.

However, the practical implementation of such analysis method poses problems because Kalman Filter implementation for real-time applications is computation-intensive in software and resource-demanding in hardware due to matrix multiplication and inversion operations.

Therefore, we have developed a design flow for design-space exploration using HW/SW Codesign to select the optimum implementation and implemented an FPGA-based cycle-accurate model of a coprocessor block for the Kalman Filter.

1.3 An Overview of the Published Research Work

The following research publications are included in the thesis and they should be read in the sequence that they appear.

1.3.1 Overview of Multiprocessor System-on-Chip Platform Modeling & Simulation

The publication included here gives an overview of the state of the art in modeling and simulation of multiprocessor system-on-chip platforms for embedded

systems and forms the basis for the research conducted and described in the second part of this thesis.

A: Kashif Virk and Jan Madsen. Computing Platforms - Multiprocessor Modeling and Simulation. The ARTIST Roadmaps for Research and Development. (Editors: Bruno Bouyssounouse and Joseph Sifakis). *Lecture Notes in Computer Science, Volume 3436, 2005.* Section 29, Pages: 388-406. Springer Scientific Publishers. Published.

1.3.2 SEND Modeling Environment

A SystemC-based System-level Modeling Framework, named SEND (**S**ystem-level Modeling Framework for **E**mbedded **N**etworked **D**evices) has been developed during the Ph.D. project. The following set of publications describe the concepts developed and the techniques employed during the progressive evolution of the modeling framework.

B: Jan Madsen, Kashif Virk and Mercury Jair Gonzalez. A System-level Multiprocessor System-on-Chip Modeling Framework. *Proceedings of the IEEE International Symposium on System-on-Chip, 2003.* (SoC'03), November 2003. Pages 147-150. Published.

C: Jan Madsen, Shankar Mahadevan, Kashif Virk and Mercury Gonzalez. Network-on-Chip Modelling for System-Level Multiprocessor Simulation. *Proceedings of the IEEE Real-Time Systems Symposium* (RTSS 2003), December 2003. Pages: 265-274. Published.

D: Kashif Virk and Jan Madsen. A System-Level Multiprocessor System-on-Chip Modeling Framework. *Proceedings of the IEEE International Symposium on System-on-Chip* (SoC'04), November 2004. Pages: 81-84. Published.

E: Kashif Virk, Knud Hansen and Jan Madsen. System-Level Modeling of Wireless Integrated Sensor Networks. *Proceedings of the IEEE International Symposium on System-on-Chip* (SoC'05), November 2005. Pages: 179-182. Published.

1.3.3 Hogthrob Sensor Network Development Platform

The Hogthrob platform architecture consists of four closely-interacting subsystems. These subsystems are: the sensing subsystem, the computing subsystem,

the communication subsystem, and the power-supply subsystem. The platform has been designed using a modular design approach and comprises one mother board (8.5cm x 7cm) which comprises the computing and the power-supply subsystems, one daughter board for the communication subsystem (4cm x 5cm) and another daughter board for the sensing subsystem. The mother board has been further divided into the analog and the digital portions with the analog portion mostly occupied by the power-supply subsystem and the computing subsystem comprising the digital portion.

F: Kashif Virk, Mohammad Shafique, Jan Madsen and Aric Menon. System-Level Modeling and Simulation of MEMS-based Sensors. *Proceedings of the IEEE International Multi-Topic Conference* (INMIC'05), December 2005. Pages: 1-6. Published.

G: Kashif Virk, Martin Leopold, Andreas Vad Lorentzen, Martin Hansen, Phillipe Bonnet and Jan Madsen. Design of a Development Platform for HW/SW Codesign of Wireless Integrated Sensor Nodes. *Proceedings of the IEEE EuroMicro Conference on Digital System Design* (DSD'05), August 2005. Pages: 254-260. Published.

H: Kashif Virk and Jan Madsen. Functional Testing of Wireless Sensor Node Designs. *Proceedings of the IEEE International Conference on Innovations in Information Technology* (Innovations'07), November 2007. Pages: 123-127. Published.

Bibliography

[1] L. Benini and G. De Micheli. Network on Chips: A New SoC Paradigm. *IEEE Computer*, 35(1):70 – 78, January 2002.

[2] David Culler, Deborah Estrin, and Mani Srivastava. Overview of Sensor Networks. *IEEE Computer*, 37(8):41–49, August 2004.

[3] David Tennenhouse. Proactive Computing. *Communications of the ACM*, 43(5):43–50, 2000.

[4] Marcus T. Schmitz, Bashir M. Al-Hashimi, and Petru Eles. *System-Level Design Techniques for Energy-Efficient Embedded Systems*. Springer, ISBN: 978-1-4020-7750-0, 2004. pp 211.

[5] G. De Micheli, R. Ernst, and W. Wolf. *Readings in Hardware/Software Co-Design*. Morgan-Kaufmann, 2001. 1st edition.

[6] The Hogthrob Project. http://www.hogthrob.dk.

[7] Univeristy of California, Berkeley. TinyOS Community Forum. http://www.tinyos.net.

[8] R. Want, T. Pering, and D. Tennenhouse. Comparing Autonomic and Proactive Computing. *IBM Systems Journal*, 42(1):129–135, 2003.

Chapter 2

Computing Platforms - Multiprocessor Modeling and Simulation

Kashif Virk and Jan Madsen. Computing Platforms - Multiprocessor Modeling and Simulation. The ARTIST Roadmaps for Research and Development. (Editors: Bruno Bouyssounouse and Joseph Sifakis). *Lecture Notes in Computer Science, Volume 3436, 2005.* Section 29, Pages: 388-406. Springer Scientific Publishers. Published.

Computing Platforms: Multiprocessor System - Modeling & Simulation

Kashif Virk and Jan Madsen
System-on-Chip Group
Computer Science & Engineering Section
Department of Informatics & Mathematical Modeling
Technical University of Denmark, Lyngby 2800, Denmark
email: {virk, jan}@imm.dtu.dk

***Abstract*—Models of Computing Platforms are extremely important but different models that correspond to various viewpoints must be integrated. Multiprocessor Computing Platforms can be tightly or loosely coupled and the abstraction level of their models can vary from the specification to the physical level. In addition to the abstraction level, there exist various modeling domain dichotomies (continuous or discrete, synchronous or asynchronous, event-triggered or time triggered, etc.) that all have their merits within thier respective application areas. The modeling of reactive multiprocessor systems, usually, demands simultaneous invocation of models across the various domains. Reconciliation of these models is a major research topic.**

In this paper, we present a survey of the state of the art in the modeling of multiprocessor computing platforms.

I. Motivation

As more computer systems are being integrated into system-on-chip (SoC) designs and as the interactions of concurrent (and, possibly, real-time) software with multiprocessing and distributed computing platforms becomes more complex, computer systems designers must reason about: computing platform design for programmability, the modeling aspects of software schedulers, the co-execution of hardware-like and software-like system-level behaviors, and the system-level performance impacts of hardware architectures that execute the software functionality. Many of the critical system-level design decisions are those that involve the anticipation of hardware/software interactions; as hardware is loaded with the software functionality, software is deployed onto a variety of hardware resources (architectures) and parts of a system (mixed hardware and software) must interact with the yet-to-be designed rest of the system (which may also include mixed hardware and software). Modeling systems in the large is an important trend in software and systems engineering. The purpose of modeling is to build models of software and systems which satisfy given requirements. Modeling plays a central role in systems engineering. The use of models can profitably replace experimentation on actual systems with incomparable advantages such as:

- Enhanced modifiability of the model and its parameters.
- Ease of construction by integration of models of heterogeneous components.
- Generality by using genericity, abstraction, and behavioral non-determinism
- Enhanced observability and controllability, especially, avoidance of the probe effect and of disturbances due to experimentation.
- Possibility of analysis and predictability by the application of formal methods.

Building models which faithfully represent complex systems is a non-trivial problem and a pre-requisite to the application of formal analysis techniques. Usually, modeling techniques are applied at early phases of system development and a higher level of abstraction. Nevertheless, the need of a unified view of the various lifecycle activities and of their interdependencies have motivated the so-called model-based approaches which rely heavily on the use of modeling methods and tools to provide support and guidance for system development and validation.

II. Landscape

A. Classification of Computation Platforms

Computation Platforms may be classified into transformational, interactive, reactive and proactive systems.

- **Transformational** computation platforms compute results with the input data available right from the start of the application without any timing constraints. The computed results are usable as and when required at any given instance.
- **Interactive** computation platforms operate on the environment-produced data without any timing constraints which are expected by already executing tasks. The results computed by those tasks are input to other tasks.
- **Reactive** computation platforms execute tasks that produce results at the times determined by the controlled process dynamics.
- **Proactive** computation platforms capture and may act on data without user intervention.

Computation platforms are structured in layers. They all contain operating systems for the basic management of the processor, virtual memory, interrupt handling, and communication [1].

Most of the future embedded computing platform applications are likely to be real-time applications that will run

on multiprocessor SoC's which are, essentially, distributed computing systems. In a multiprocessor or a distributed computing platform, the processing elements can be connected through shared memory, dedicated communication links or a communication network [2].

A major dividing line inevitably exists between the *discrete* embedded computing platforms and the, essentially, *continuous* physical environments in which they are embedded. The discrete embedded computing platforms (comprising both the hardware and the software), in turn, contribute to a number of additional dichotomies at the stage of their mathematical modeling.

The early embedded computing platforms were, essentially, *sequential* computing platforms but, as they are extended and become more complex, a need for the concepts of hierarchy and information sharing between their sub-systems arises (as in *concurrent* systems). To mathematically characterize these concepts, a global notion of a computation step is considered. Thus, the dichotomy between the *synchronous* and the *asynchronous* computation models appears. Moreover, to model the behavior of an embedded computing platform in response to changes in inputs (as in reactive systems) can be described in either an *event-triggered* or a *time-triggered* fashion. In addition, timeliness can be a central issue apart from the correct functioning (as in *real-time* systems) requiring the explicit inclusion of time in the computation model.

Furthermore, the application domains contribute additional modeling preferences to the discrete embedded computing platforms. A major such division is between the control-oriented applications, leading to *control-flow* or state-based computation model (where the complexity arises due to the massive numbers of control locations in a computation), and data-oriented applications, leading to *data-flow* computation model (where there is much structure in the data on which a large number of operations can be performed in a few control locations). Most of the discrete embedded computing platforms are composed of sub-systems that are designed according to some or all of the various types of computation models mentioned above. Therefore, some or all of the above-mentioned dichotomies have to be reconciled in the same discrete embedded computation platform using appropriate meta-models. That is, the embedded computing platforms composed of continuous/discrete, sequential/concurrent, synchronous/asynchronous, state-based/data-flow, event-triggered/time-triggered, real-time/non real-time components have to be methodically developed based on well-defined underlying semantics.

In addition, no matter how the individual sub-systems are modeled and analyzed on their own, eventually, the composed system has to be subject to analysis to ascertain that the system exhibits the desired behaviors only in a physical environment. Thus, a very natural way to model an embedded computing platform is by including the elements of the continuous state and the discrete state in the same *hybrid* computation model.

Another major challenge is to combine the existing analysis techniques from the various paradigms and to devise a coherent verification methodology for multi-paradigm systems. In particular, to ascertain which aspects of the analysis benefit from the existing capabilities of each paradigm.

B. Models of Concurrent Systems (Parallel & Distributed Computing Platforms)

Despite an apparent trend towards parallel computers being composed of nodes of independent processor-memory pairs connected by some interconnection network, it is by no means certain that there is a definite progression towards a single class of parallel architectures. Instead, there are numerous classes of parallel architectures. Similarly, there are numerous models of parallel computation, some specifically suited to particular architecture classes, while others are suitable across a range of parallel architecture classes.

Models of parallel computation are required to act as a map between disparate programming languages and disparate architectures. Hence, an application developed according to the model is executable on the various architectures and its performance is predictable.

A model is said to be architecture-independent if it is general enough to model a range of architecture types. So, the application source code is portable to various parallel architecture classes without modification.

There are several levels at which a model of computation may exist:

- **Specification Level:** at the specification level, the model of computation provides an unambiguous description of a computational problem without any notions of its execution or implementation. Typical examples are:
  - *State Transition Models* (e.g., FSM's, CFSM's, Petri Nets, Process Algebras, Duration Calculus, π-Calculus, etc.)
  - *Data Flow Models* (e.g., {Kahn} Process Networks, Data Flow Graphs, Synchronous Data Flow Graphs, etc.)
  - *Discrete Event Models* (e.g., HDL Simulators, etc.)
- **Performance Level:** at the performance level, the model provides a basis for the solution of a computational problem. Thus, it forms the basis for the design, discussion and prediction of the performance of algorithms. The most common examples of such models are: Turing Machines, RAM, PRAM, BSP, LogP, etc.
- **Programming Level:** at the programming level, the model provides a precise, high-level description of correct and efficient methods for the solution of the particular computational problem, e.g., Imperative Programming, Declarative Programming (Applicative Programming - Functional Programming, Predicative Programming - Logic Programming), etc.
  - *Communication Sub Model:* Communication is, probably, the most important aspect of a computation model. Therefore, in any model of computation, communication needs to be accurately accounted for. The most common communication abstractions are: MPI/PVM, OpenMP, IPC, RPC, TCP/IP, OSI, etc.

- **Architectural Level:** at the architecture level, the model describes the characteristics of a real machine on which the computational problems will be implemented and solved, e.g., SISD (von Neumann, Harvard, etc.), SIMD (vector, array, etc.), MISD (systolic, etc.), MIMD (Parallel - Shared Memory, Distributed Memory, etc. Distributed - Clusters of Workstations, Grids, etc.), Data Flow, Reduction, Neural Network, etc. [3].
  - *Network Sub Model:* The two basic measures of network models are latency and bandwidth which determine the network geometry. The most commonly modelled network topologies are Hypercube, Butterfly, Torus, Mesh, etc.

A multiprocessor system is *tightly coupled* so that the global status and workload information on all processors can be kept current at a low cost. The system may use a centralized scheduler. When each processor has its own scheduler, the decisions and actions of the schedulers of all the processors are coherent. In contrast, a distributed system is *loosely coupled*. In such a system, it is costly to keep the global status and workload information current. The schedulers on different processors may make scheduling and resource access control decisions independently. As a consequence, their decisions may be incoherent as a whole. In modeling distributed systems, the operating systems have a major role. Moreover, in a distributed system, if the processors can be used interchangeably, they are identical and if a message from a source processor to a destination processor can be sent on any of the links connecting them, then the links are identical as well. In contrast, processors of different types cannot be used interchangeably. Different types of processors may either be functionally different or they may be of different types for many other reasons. A computation platform comprising such processors, which are loosely coupled, is called a distributed heterogeneous system [2].

C. Models of Reactive Systems

Reactive computing systems continuously interact with their environment. These systems are, in general, composed of concurrent, interacting sub-systems or processes which may cooperate, synchronize, and share resources. It is the role of a scheduler to coordinate the execution of system activities in order to guarantee a correct functioning of the system.

1) Control-flow vs. Data-flow Models: The family of formal languages known as synchronous languages have shown that they are simple enough to appeal to the engineering community and expressive enough to model non-trivial applications in embedded control. *Lustre* and *Signal* have a data-flow (declarative) style whereas *Esterel* and *Statecharts* are considered as control-flow or state-based (imperative). Each language comes with a bunch of analysis techniques and well-developed toolboxes. One of the major benefits of *Signal*, *Lustre*, and *Esterel* is the clearly-documented formal semantics which acts as a description of a meta-model. The clock calculus in *Lustre* and *Signal* and the constructive semantics of *Esterel*, for example, can be used for the static checking of the desired properties of an instance (an application model) based on the formal semantics of the languages the defined correctness criteria. Major such properties are the determinism in a controller and the causal consistency at every macro (computation) step. The **Statemate** tool based on *Statecharts* checks the type-coherence of the variables in a model and performs some simple consistency checks.

These tools are finding their ways into modeling the digital components of several embedded applications such as power and digital signal processing systems (*Signal*), electronic design automation and aerospace systems (*Esterel*), and railway and aerospace systems (*Lustre*). These tools also provide efficient automatic code generation mechanisms. Thus, after the compilation stage, the design can be subjected to further formal verification and code optimization, eventually, leading to automatically-generated controller code (*C*, *Ada*, or *VHDL*).

Statecharts has had its original popularity in the aerospace sector but it is gaining popularity for the embedded system design due to inclusion into the *UML* family of languages. The tool **Rhapsody**, though no longer in the framework of synchronous languages, is a valuable tool for modeling object-oriented distributed embedded systems.

All of the above-mentioned tools, however, have so far been applied on an individual basis in the respective applications. Considering the growing needs of multi-paradigm modeling, two European projects have been exploring the combination potentials of these tools SACRES for combining *Signal* and *Statecharts*, and SYRF for the combination of *Signal*, *Lustre*, and *Esterel*. The work in SACRES has resulted in relating synchrony with asynchrony and the conditions under which these paradigms can be combined. The work in SYRF has resulted in the development of cross-compilation tools for *Lustre*, *Signal*, and *Esterel* (loose integration), an environment for the multi-paradigm modeling (tight integration), and code distribution for embedded systems.

2) Event-triggered vs. Time-triggered Models: As described above, each member of the synchronous language family has been extensively used for the design of embedded systems. A recent activity has been to combine the analysis of continuous systems (as modeled in **Matlab**) with the meta-model verification and efficient code generation capabilities of the *Signal* environment. This is one of the approaches in a series of attempts at the problem of the analysis of hybrid systems.

In recent years, **Matlab** has been extended with a modeling facility for describing a discrete controller (*Stateflow* - with a syntax reminiscent of *Statecharts*). However, the underlying computation mechanism for the simulation of the discrete part of a model is the same as the continuous part of the model. That is, all signals are defined over continuous time and the simulation is time-triggered based on the lowest sample period.

3) Synchronous vs. Asynchronous Models: As discussed above, not all applications can, naturally, be modeled as a globally- synchronous system. A recent development has been to relate the notions of synchrony and asynchrony in the context of data-flow languages (in particular, *Signal*). This work introduces the theoretical notions that can be used to

characterize an asynchronous network of locally-synchronous nodes and the compositionality properties (as a meta-model property in this context). Similar ideas are developed in the context of imperative languages where it is shown how constructively-checked *Esterel* can be used as an input language to the **Polis** environment, compiling into co-design finite state machines communicating over one-place buffers.

4) Continuous vs. Discrete Models: Recent years have seen the extension of the application of formal methods to the models with both the continuous and the discrete elements. A typical goal of verification is to show that an invariance holds over a model. In particular, a bad property does not hold in any reachable state of a system. Since digital controllers are increasingly complex with mode changes and multiple inputs and outputs, and the goal of the controller is, typically, to avoid a bad state in the physical environment, the traditional methods for proving the invariance are not applicable (neither the computing science methods for proving the properties of discrete systems, nor the control theory methods for the analysis of continuous systems). Several techniques for dealing with this inherently difficult problem have been proposed.

D. Models of Real-Time Systems

Real-time computing platforms are the systems whose correctness depends on the respect of timing constraints. Although real-time systems have become ubiquitous by now, their design still poses challenging problems and is a very active domain of research. Real-time systems have to reconcile functional, physical, and timing requirements that are often antinomic.

Currently, the validation of real-time systems is done by experimentation and measurement on specific platforms in order to adjust design parameters and, hopefully, achieve conformity to QoS requirements. The existence of modeling techniques for real-time systems is a basis for rigorous design and should drastically ease their validation. Modeling a real-time system should allow to validate its design before implementing the system, and to prove its correctness using formal methods. For reactive real-time systems, it is important to build models that faithfully represent their behavior. In such models, the application has to be modeled together with the behavior of its environment and dynamics [4].

A modeling framework accompanying the design process of real-time systems and providing a methodology, can guide and accelerate the design process, replace ad hoc solutions by standard constructions, and improve the quality of the model. For a modeling framework to be useful, it should meet the following requirements:

- It should be sufficiently general to allow, in a natural and comprehensive way, the specification of resource contention, synchronization, priority selection, urgency, preemption, periodic, aperiodic, and sporadic processes, and various scheduling disciplines on uni- or multiprocessor systems.
- It should be based, despite of their expressiveness, on an analyzable and executable model. That is, it should be operational rather than descriptive, so as to reduce risks of errors caused by passing from one formalism to another.
- It should be founded on theoretical results ensuring well-defined semantics, supporting a modular specification, compositionality, and allowing, to some extent, correctness by construction.
- It should be practical and applicable. That is, it should provide an intuitive, high-level modeling formalism, together with a design methodology, and guidelines or standard constructions for common problems. Moreover, it should allow feasible algorithms for automatic analysis, supporting the design process, and be supported by tools.
- It should help detecting design errors by providing diagnostics at an early stage allowing debugging of the design or gain confidence in its correctness and support a predictable model, in the sense that unexpected interaction between separately modeled behavioral requirements is ruled out as far as possible.

Existing formalisms and tools are designed to meet different subsets of the requirements mentioned above. However, as some of the items seem difficult to reconcile - for example, the generality of the model and the support for an early detection of design errors - they are not equally addressed by one framework [5].

Component-based engineering is of paramount importance for rigorous system design methodologies. It is founded on a paradigm which is common to all engineering disciplines: complex systems can be obtained by assembling components (building blocks). Components are, usually, characterized by abstractions that ignore implementation details and describe properties relevant to their composition, e.g., transfer functions, interfaces. Composition is used to build complex components from simpler ones. It can be formalized as an operation that takes in components and their integration constraints. From these, it provides the description of a new, more complex component.

Component-based engineering is widely used in VLSI circuit design methodologies, supported by a large number of tools. Software and system component-based techniques have known significant development, especially, due to the use of object technologies supported by languages such as *C++*, *Java*, and standards such as *UML* and *CORBA*. However, these techniques have not yet achieved the same level of maturity as has been the case for hardware.

1) Scheduling Theory-based Approaches: Well-established scheduling theory and scheduling algorithms have been successfully applied to real-time systems development. Schedulability analysis essentially consists in checking that the system meets the schedulability criteria prescribed by the theory, which allows efficient schedulability analysis tools. It does not require the use of a model representing the dynamic behaviour of the system to be scheduled. Current engineering practice, essentially, adopts this approach.

Existing scheduling theory requires the application to be set into the mathematical framework of the schedulability criterion. Studies to relax such hypotheses have been carried

out. However, most of these schedulability results apply only for particular process models or do not allow complex interaction between the components such as shared resources apart from the processor, atomicity, or communication. Generally, functional and timing properties are specified and verified separately and no unified approach for general scheduling problems has been proposed so far.

2) Model-based Approaches: To overcome these limitations, an alternative approach consists in building, explicitly, a timed computation model of the real-time application, that is, the application processes together with their possible interaction, and verifying schedulability [6] or extracting a scheduler [7], without considering the particular scheduling policies. Modeling methodologies and tools for real-time systems have shifted into the focus of research in the recent years.

The controller synthesis paradigm for discrete-event systems [8] and timed systems [9], [10], [11], [12] provides a general framework for scheduling. This is the most general approach but the algorithmic method for synthesizing a controller is of prohibitive complexity. For this reason, sometimes, the existence of an invariant implying satisfaction of the timing constraints is explored, using real-time verification techniques [13], [14], [15], [16], [17], [18] and tools such as **Kronos** [19], [20], **Uppaal** [21], [22], **Verus** [23], **Cospan** [24], or **HyTech** [25]. A non-empty invariant satisfying the timing constraints is a sufficient condition for schedulability, requiring techniques of lower complexity than synthesis which do not distinguish between controllable and uncontrollable actions.

There are several other approaches to tackle the complexity of verifying real-time systems, or synthesizing schedulers. For example, [26] discusses incremental verification of communicating Time Petri Nets, based on assume-guarantee reasoning. [27] presents a scheduler synthesis tool based on constraint satisfaction for a simple process model that nevertheless allows shared resources, and a timing specification in Real-Time Logic. [28] discusses the analysis of non-deterministic real-time systems using the (max; +) algebra, which does not require exploring the state space like traditional model-checking techniques. [29] provides an algorithm synthesizing a programmable logic controller from a specification described by a fragment of the duration calculus. [30] describes a formal low-level framework for real- time system models, where processes are described by sets of possible behaviors. This framework is intended as a unifying meta-model rather than to directly model real- time applications. [31] discusses modeling and verification of preemptive real-time systems with hybrid automata. Similarly, [32] describes a methodology for modeling a general class of real-time systems with resource constraints, synchronization and context switching overhead, and atomicity of code segments, as hybrid systems. The method is applied to the timing analysis of Ada programs. Adopting the same framework, [33] discusses the timing analysis of partially implemented systems, where lacking pieces of code are specified in Graphical Interval Logic. [34], [35] discuss a formal model of the *Ravenscar* [36] subset of *Ada 95*, allowing to verify applications using the model-checker **Uppaal**.

3) Meta-Model-based Approaches: Among the modeling and design tools, we shall mention the **Ptolemy** [37] project and toolset aiming at heterogeneous modeling, simulation, and design of embedded systems by integrating different models of computation. Another tool for the integration of heterogeneous models is the **SPI Workbench** [38], which uses graphs of communicating processes annotated with timing intervals, as a unifying abstract representation serving as a basis for verification and hardware/software co- design. **Giotto** [39] is a tool-supported design methodology for distributed embedded systems based on the time-triggered paradigm. It consists of a programming language, and a platform-dependent part including a compiler and a runtime library. Taxys [40], [41] is a tool used for the development and verification of embedded systems in the telecommunication domain. The system and its environment are specified in the synchronous language *Esterel* [42] annotated with timing constraints. The model can be verified by the model-checker **Kronos**, and compiled to *C* code by the *Esterel* compiler **Saxo-RT** [43].

4) Process Algebra-based Approaches: There is some work aiming at integrating model-based analysis of real-time systems, and scheduling theory. The interest of considering particular scheduling policies in a model-based approach is twofold. First, it allows to verify both the functional correctness, and the timeliness, of a scheduled real-time system, whereas the same system without a scheduler, generally does not meet its timing constraints. Second, restricting the set of possible behaviors helps to manage the state explosion problem. Most of this work is based on process algebras extended with a notion of priority. [44] defines a process algebra based on *CCS* (Calculus of Communicating Systems) [45] with real-time semantics and dynamic priorities. In the process algebra *RTSL* (Real-Time Specification Language) [46], scheduling policies such as RMS (Rate-Monotonic Scheduling) and EDF (Earliest Deadline First) can be modeled by a function associating, with any system state, a subset of processes that remain enabled after priority choice. The process algebra *ACSR* (Algebra of Communicating Shared Resources) [47], [48] provides a framework with discrete and dense-time semantics for modeling coordination between processes including shared resources, synchronization, preemption, static priorities, and exception handling. A prioritized strong bisimulation ensures compositionality. The **Paragon** toolset [49] for the specification and verification of real-time systems is based on *ACSR*. The system can be modeled in a graphical specification language. Verification is done by state-space exploration, or checking for bisimulation with a process specifying a high-level behavior. [50] discusses the modeling of real-time schedulers in *ACSR-VP*, an extension of *ACSR* with value passing communication. Schedulability analysis amounts to symbolically checking the, possibly, parameterized model for bisimulation with a non-blocking process, and synthesizes the parameter values for which the system is schedulable. In [51], models of basic process specifications are given, and schedulers for EDF and the priority inheritance protocol [52] are modeled under *ACSR-*

VP. [53] presents a modeling methodology for fault-tolerant distributed real-time systems. Processes and fault models are specified in a process algebra based on Timed *CCS*; liveness properties and deadlines are expressed in a logic based on *Modal Timed-Calculus*. The authors give examples of a best-effort EDF scheduler, and a planning-based scheduler where processes are only scheduled if their deadlines are guaranteed to be met. [54] model real-time processes scheduled under EDF as timed automata, and model-check the obtained representation using **Uppaal**. However, their modeling method is not compositional. [55] introduces I/O timed components, essentially, timed automata with an interface declaration, as a modeling formalism guaranteeing non-zeno and non-blocking synchronization by construction. Information about the interface of I/O timed components is used by a relevance calculus to make abstraction from components that are irrelevant for proving a given property specified as an observer process. **MetaH** [56] is a development tool initially designed for avionics applications. It accompanies the development process of real-time systems from specification down to code generation, and implements schedulability analysis based on the results of [57], [58] extending rate-monotonic analysis. It is also possible to specify error models, and carry out reliability analysis. Sometimes, a deductive approach is used to verify correctness of a scheduler [59], [60], [61] using theorem provers. In [60], real-time programs with timing constraints, fault models, and scheduling policies are modeled in the logic *TLA* (Temporal Logic of Actions) [62]. Proving that scheduling the real-time system under a certain discipline, both specified in *TLA*, is feasible, amounts to verifying a schedulability condition similar to the results from scheduling theory.

E. Application Domains of Computation Platforms

1) Networks-on-Chip: In the modern silicon technologies, with minimum device geometries in the nanometer range (<100nm), the on-chip interconnection fabric is a major source of delay and power consumption which is challenging the on-chip communication infrastructure and forcing a change from device-centric to interconnect-centric design methodologies.

A Network-on-Chip (NoC) is a disciplined approach to replace the current ad hoc wiring of the IP blocks that pairs scalable communication performance and minimal interconnect cost. It separates the computation from communication by allowing the computational blocks to communicate with one another via a uniform interface. A NoC can be based on packet switching communication to flexibly share link capacity between either homogeneous or heterogeneous network clients and to provide multiple communication services over a uniform infrastructure with fixed topology.

An efficient combination of the best-effort and the guaranteed services in a NoC is a challenge [63]. The other key challenges for designing NoCs include automated synthesis [64], [65], low-power [66], [67], verification and testing [68], [69], and fault-tolerance [70].

In order to address these challenges, accurate modeling of the systems and all the interrelationships among the diverse processors, software processes and physical interfaces and interconnections, is needed. One of the primary goals of the system-level modeling for networks-on-chip is to formulate a modeling framework within which a broad class of designs can be developed and explored.

In addition, to support the designers of single-chip based embedded systems, which includes multiprocessor platforms running dedicated real-time operating systems (RTOS's) as well as the effects of on-chip interconnect network, a system-level modeling/simulation environment is required to support an analysis of the:

- consequences of different mappings of tasks to processors (software or hardware),
- network performance under different traffic and load conditions,
- effects of different RTOS selections, including various scheduling, synchronization and resource allocation policies.

The traditional network models like OPNET [71] are not suited for NoC's, since they model only the abstract communication structure without any support for chip-level architecture modeling. In [72], [73], the concept of on-chip, packet-switched micro-networks has been introduced that borrows ideas from the layered design methodology for data networks. The work on the system-level exploration of the communication architecture can be subdivided into static analysis models [74], [75] and simulation-based models [76], [77]. Lahiri et al. [78] have proposed a hybrid model combining simulation with analytical post-processing to achieve higher accuracy of the performance estimation. The SystemC Open Core Protocol (SOCP) communication channel in the StepNP simulation model [79] addresses the exploration of the communication infrastructure based on the OCP semantics. Serge Goosens et al. [80] further abstract from architecture-specific communication primitives to establish a unified modeling framework for the investigation of heterogeneous on-chip networks. The NoC modeling framework proposed in [81] deals with generalized abstract tasks, processing elements, and communication infrastructures instead of dealing with each specific application and system architecture. This not only broadens the applicability of the modeling framework, but also leads to a better understanding of the problem at hand.

The current NoC modeling approaches do not cope with the requirements introduced by the system-level design of full-fledged on-chip networks. In order to apply analytical models, enhanced algorithms are necessary to model the performance of complex network topologies with sophisticated arbitration mechanisms. Equally, current NoC simulation models fall short to provide efficient support for the exploration of on-chip networks.

2) Wireless Sensor Networks: The recent advances in low-power embedded processors, radios, and micro-electromechanical systems (MEMS) have made possible the development of networks of wirelessly interconnected sensors. The new computing paradigm enabled by the ad hoc wireless sensor networks will be a key in making computation more

proactive. The silicon-based wireless sensors and the ad hoc sensor networks represent exciting new technologies with broad societal impacts and a wide range of new commercial opportunities. As the wireless sensor technology continues to advance, one day, it will be possible to have these compact, low-cost wireless sensors embedded throughout the environment, in homes, offices, and ultimately inside people. With the continued advances in power management, these systems should find more numerous and more impressive applications. Until that day, there is a rich set of research problems associated with the distributed wireless sensors that require very different solutions than the traditional sensors and multimedia devices [82].

With their focus on the applications requiring a tight coupling with the physical world, as opposed to the personal communication focus of conventional wireless networks, the wireless sensor networks pose significantly different design, implementation, and deployment challenges. Their application-specific nature, severe resource limitations, long network life requirements, and the presence of sensors lead to an interesting interplay between sensing, communication, power consumption, and topology that the designers need to consider. Energy dissipation, scalability, and latency must all be considered in designing network protocols for collaboration and information sharing, system partitioning, and low-power electronics design [83].

The existing tools for modeling wireless networks focus only on the communication problem and do not support the modeling of power and sensing aspects that are essential to the design of wireless sensor networks. A model of computation is of prime importance as a clean starting point for the synthesis of modern computing platforms. The wireless sensor networks will not only require new models of computation, but also new models of the physical world.

In the design automation domain, synthesis of the nodes for the wireless sensor networks will pose a number of new problems. Moreover, debugging and verification are the most expensive and time-consuming components in the modern design flow. Due to the heterogeneous nature and the complex interaction between the components, it is expected that the same will be true for the nodes of the wireless sensor networks. In particular, the techniques for error and fault detection and testing collaboration will be of prime importance.

Middleware will be in strong demand to enable the development of new applications. Tasks such as sensor data filtering, data compression, data fusion, data searching and profiling, exposure coverage, and tracking will be ubiquitous. It is expected that new tasks will be defined and accomplished, for example, sensor allocation and selection, sensor positioning, sensor assignment and efficient techniques for the sensor data storage [84].

In the software domain, main emphasis will be on the RTOS's (Real-Time Operating Systems). There is a need for an ultra-aggressive, low-power management due to the energy constraints and a need for comprehensive resource accounting due to the demands for privacy and security and, in a number of cases, the support for mobility-related functions as well. There is also a need for the overall energy consumption balanced architectures. Another issue is the wireless sensor organization and the development of interfaces between the components. Finally, due to the privacy, security, and authentication concerns, techniques like unique IDs for the CPU and other components can be of high importance.

III. Assessment

In order for a high-level modeling environment to be effective for design exploration, it must be abstract, or high enough to enable rapid design trade-offs, but detailed enough to include a time basis for performance modeling.

The development of a general theoretical modeling framework for component-based engineering is one of the few grand challenges in information sciences and technologies. The lack of such a framework is the main obstacle to mastering the complexity of heterogeneous systems. It seriously limits the current state of the practice, as attested by the lack of development platforms consistently integrating design activities and the often prohibitive cost of validation.

A major factor limiting the use of parallel computing platforms in the mainstream computing is the lack of general-purpose parallel computation models. Moreover, some specialists who believe that finding a unifying computation model is just not possible have gone in another direction, developing parallel software that lacks portability. On the software side, the architecture differences in the parallel computing platforms correspond to a large set of different parallel models and languages often architecture-dependent and that offer only partial solutions to programming portable parallel applications in sequential computing using standard languages like *C*, *Pascal*, and *Fortran*. Many parallel programming languages used today are of the low-level variety which require the programmer to face the architectural issues of the parallel computing platform on which the application executes.

On the other hand, high-level parallel languages abstract from architectural issues but deliver unpredictable performance on different architectures. Thus, porting the same program to different parallel computation platforms from, say, a message-passing multi-computer to a shared-memory multiprocessor can dramatically alter the platform's performance.

Existing component technologies encompass a restricted number of interaction types and execution models, for instance, interaction by method calls under asynchronous execution. We lack concepts and tools allowing integration of synchronous and asynchronous components, as well as different interaction mechanisms, such as communication via shared variables, signals, rendezvous. This is essential for modern systems engineering, where applications are initially developed as systems of interacting components, from which implementations are derived as the result of a co-design analysis.

The application of component-based design techniques raises two strongly related and hard problems. First, the development of a theory for building complex heterogeneous

systems. Heterogeneity is in the different types of component interaction, such as strict (blocking) or non strict, data driven or event driven, atomic or non atomic and in the different execution models, such as synchronous or asynchronous. Second, the development of theory for building systems which are correct by construction, especially with respect to essential and generic properties such as deadlock-freedom or progress. In practical terms, this means that the theory supplies rules for reasoning on the structure of a system and for ensuring that such properties hold globally under some assumptions about its constituents e.g. components, connectors. Tractable correctness by construction results can provide significant guidance in the design process. Their lack leaves a posteriori verification of the designed system as the only means to ensure its correctness (with the well-known limitations).

Co-design for system-level modeling has been limited by the view that all computation should be restricted to the reactive system models - mathematical models of computation unified by the event or token-based foundations. The resulting executable specifications are designed to respond to testbench-style inputs that model the external environment in which the system is intended to operate. The presumptions are that the computer system being designed is passive and it should be isolated from its operating environment.

IV. Trends

Finding solutions to the problems and limitations in parallel computation requires two actions:

- Make the design and implementation of general-purpose parallel computing platforms capable of supporting a wide range of programming models and providing predictable performance.
- Make the definition of programming models architecture-independent, allowing abstraction and portability across different parallel computing platforms. At the same time, make these models simple and expressive.

An important step to success is the definition of high-level, architecture-independent languages to demonstrate that parallel programming is no more difficult than sequential programming.

Low-level approaches, such as Parallel Virtual Machine (PVM) and Message Passing Interface (MPI), are driven by heterogeneous parallel computing, which tries to offer, on different computers, library primitives for parallelism and communication. These approaches partly meet the portability goal but are based on tedious low-level library functions and do not free the programmer from the issues of concurrency, communication, and synchronization. In fact, even though the PVM and the MPI are the de facto standards in parallel programming, their related programming style looks in many respects like the assembler-level programming in sequential computing.

However, several proposed high-level approaches the Bulk Synchronous Parallel (BSP), the LogP, and the Bird-Meertens Formalism may represent good candidates for architecture-independent programming models on general-purpose computers.

Other promising models are the skeleton-based and the actor-based languages. Although these models suffer from low performance, they represent an interesting starting point toward architecture-independence because they abstract from architectural issues and allow predictable performance. If the parallel programming community convinces itself that it needs a clear strategy based on high-level languages to find a unifying model for parallel computation, these models can be used to drive this process. Adopting this strategy would unite high-level programming, generality, and high-performance, leading parallel computation to the computing mainstream [85].

Increasingly, the operating environment of a computer system is another computer system. Accordingly, next generation computer system modeling must be based not only on the reaction of a passive computer system to its operating environment, but upon the active cooperation and coordination sharing across model boundaries such as resources. Computer system designers must be able to capture the sharing effects or the anticipated interactions of concurrent software executing on multiple hardware resources over a range of design variations. More than understanding the response of the system, this is about understanding the response of the design.

Searching a complex design space for designs that satisfy performance criteria can be thought of as isolating and analyzing the prevalent performance models that arise between the corner cases in a design space. To fully analyze a computer system, the designers must isolate these prevalent performance models and the ranges over which they are valid. A designer can then understand the effects of software loading, resource variations, and resource sharing [86].

A grand unified approach to modeling computing platforms systems would seek a modeling framework that serves all purposes. One approach is to create the union of all the frameworks, which have been proposed so far, providing all of their services in one bundle. But the resulting framework would be extremely complex and difficult to use, and designing and synthesis and validation tools would be difficult. A more feasible alternative is to choose one concurrent framework and show that all the others are the special cases of that. This is relatively easy to do in theory. Most of these frameworks are sufficiently expressive to subsume most of the others. The disadvantage is that this approach does not acknowledge each models strengths and weaknesses. A final alternative is to mix frameworks either heterogeneously but instead of forming the union of their services, preserve their distinct identity, or hierarchically where a component in one framework is actually an aggregate of components in another [87].

These are but a few of the interesting research problems for modeling computation platforms for embedded systems. There are many more. Modeling Configurable Computation Platforms offers interesting opportunities and challenges and potentially relates strongly to the problem of selecting appropriate computational models.

As mentioned above, there are also interesting and challenging problems in the modeling of networks, particularly providing quality-of-service guarantees in the face of unreliable resources. Finally, models are required to develop appropriate hardware and software design techniques that minimize power consumption which are critical for portable devices and wireless microsensor networks.

ACKNOWLEDGEMENTS

This work is a joint first author effort. The cooperation and support provided by the department of Informatics and Mathematical Modeling (IMM) and ARTIST is gratefully acknowledged.

REFERENCES

[1] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri, *Scheduling in Real-Time Systems*. John-Wiley & Sons, 2002.

[2] J. W. S. Liu, *Real-Time Systems*. Prentice-Hall, 2000.

[3] D. K. G. Campbell, "A Survey of Models of Parallel Computation," University of York," pp. 1-37, March 19 1998.

[4] J. Sifakis, "Modeling Real-Time Systems - Challenges and Work Directions," in *EMSOFT, Lecture Notes in Computer Science 2211*, October 2001.

[5] Gregor Gossler, "Compositional Modeling of Real-Time Systems - Theory and Practice," Ph.D. Thesis, Universite Joseph Fourier, Grenoble, France, pp. 1-141, September 2001.

[6] V. Bertin, M. Poize, and J. Sifakis, "Towards Validated Real-Time Software," in *Proceedings of the 12th EuroMicro Conference on Real-Time Systems*, 2000, pp. 157–164.

[7] P. Niebert and S. Yovine, "Computing Optimal Operation Schemes for Chemical Plants in Multi-Batch Mode," *Proceedings of Hybrid Systems, Computation and Control - Lecture Notes in Computer Science, Springer-Verlag*, vol. 1790, 2000.

[8] P. J. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete-Event Processes," *SIAM Journal of Control and Optimization*, vol. 1, no. 25, 1987.

[9] E. Asarin, O. Maler, and A. Pnueli, "Symbolic Controller Synthesis for Discrete and Timed Systems," *Proceedings of Hybrid Systems II - Lecture Notes in Computer Science, Springer-Verlag*, vol. 999, 1995.

[10] O. Maler, A. Pnueli, and J. Sifakis, "On the Synthesis of Discrete Controllers for Timed Systems," *STACS'95 - Lecture Notes in Computer Science, Springer-Verlag, E. W. Mayr and C. Puech (Editors)*, vol. 900, pp. 229–242, 1995.

[11] K. Altisen, "Generation Automatique d'Ordonnancements des Systemes Temporises," M.Sc. Thesis, ENSIMAG, INPG, Grenoble, France, 1998.

[12] S. Tripakis and K. Altisen, "On the Controller Synthesis for Discrete and Dense-Time Systems," *Proceedings of Formal Methods '99 - Lecture Notes in Computer Science, Springer-Verlag, J. M. Wing, J. Woodcock and J. Davies (Editors)*, vol. 1708, pp. 223–252, 1999.

[13] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic Model Checking for Real-Time Systems," *Proceedings of 7th Symposium on Logics in Computer Science (LICS'92) and Information Computation*, vol. 111, pp. 193–244, 1994.

[14] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The Algorithmic Analysis of Hybrid Systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3–34, 1995.

[15] A. Bouajjani, S. Tripakis, and S. Yovine, "On-the-Fly Symbolic Model Checking for Real-Time Systems," in *Proceedings of the IEEE Real-Time Systems Symposium, IEEE Computer Science Press*, 1997.

[16] H.-H. Kwak, I. Lee, A. Philippou, J.-Y. Choi, and O. Sokolsky, "Symbolic Schedulability Analysis of Real-Time Systems," in *Proceedings of the IEEE Real-Time Systems Symposium, IEEE Computer Science Press*, 1998, pp. 409–418.

[17] S. Tripakis, "L'Analyse Formelle des Systemes Temporises en Pratique," Ph.D. Thesis, Universite Joseph Fourier, Grenoble, France, 1998.

[18] P. Pettersson, "Modeling and Verification of Real-Time Systems using Timed Automata - Theory and Practice," Ph.D. Thesis, Uppsala University, Sweden, 1999.

[19] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, "The Tool Kronos," *Hybrid Systems III, Verification and Control, LNCS, Springer-Verlag*, vol. 1066, pp. 208–219, 1996.

[20] S. Yovine, "KRONOS: A Verification Tool for Real-Time Systems," *Software Tools for Technology Transfer*, vol. 1, no. 1+2, pp. 123–133, 1997.

[21] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a Nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[22] H. Jensen, K. Larsen, and A. Skou, "Scaling up Uppaal: Automatic Verification of Real-Time Systems Using Compositionality and Abstraction," in *Proceedings of the FTRTFT 2000, LNCS, Springer-Verlag*, 2000.

[23] S. Campos, E. Clarke, W. Marrero, and M. Minea, "Verus: a Tool for Quantitative Analysis of Finite-State Real-Time Systems," in *Proceedings of the Workshop on Languages, Compilers and Tools for Real-Time Systems*, 1995.

[24] R. H. Hardin, Z. Har'El, and R. P. Kurshan, "COSPAN," in *Proceedings of the CAV'96, R. Alur and T. A. Henzinger (Editors), LNCS, Springer-Verlag*, vol. 1102, 1996, pp. 423–427.

[25] T. Henzinger, P. Ho, and H. Wong-Toi, "HyTech: A Model Checker for Hybrid Systems," *Software Tools for Technology Transfer*, pp. 110–122, 1997.

[26] G. Bucci and E. Vicario, "Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets," *IEEE Transactions on Software Engineering*, vol. 21, no. 12, 1995.

[27] A. K. Mok, D.-C. Tsou, and R. C. M. de Rooij, "The MSP.RTL Real-Time Scheduler Synthesis Tool," pp. 118–128, 1996.

[28] G. P. Brat and V. K. Garg, "Analyzing Non-Deterministic Real-Time Systems with (max,+) Algebra," in *Proceedings of the RTSS'98*, 1998, pp. 210–219.

[29] H. Dierks, "Synthesizing Controllers from Real-Time Specifications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 1, pp. 33–43, 1999.

[30] E. A. Lee, "Modeling Concurrent Real-Time Processes using Discrete Events," *Annals of Software Engineering, Special Volume on Real-Time Software Engineering*, 1998.

[31] S. Vestal, "Modeling and Verification of Real-Time Software using Extended Linear Hybrid Automata," Honeywell Technology Center, Tech. Rep., 1999.

[32] J. C. Corbett, "Timing Analysis of Ada Tasking Programs," *IEEE Transactions on Software Engineering*, vol. 22, no. 7, 1996.

[33] G. S. Avrunin, J. C. Corbett, and L. K. Dillon, "Analyzing Partially-Implemented Real-Time Systems," *IEEE Transactions on Software Engineering*, vol. 24, no. 8, 1998.

[34] K. Lundqvist and L. Asplund, "A Formal Model of a Run-Time Kernel for Ravenscar," in *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications RTCSA'99*, 1999, pp. 504–507.

[35] K. Lundqvist, L. Asplund, and S. Michell, "A Formal Model of the Ada Ravenscar Tasking Profile; Protected Objects," in *Proceedings of Ada-Europe '99, LNCS, Springer-Verlag*, 1999.

[36] B. Dobbing and A. Burns, "The Ravenscar Tasking Profile for High-Integrity Real-Time Programs," in *Proceedings of the ACM SigAda Annual Conference*. ACM Press, 1998, pp. 1–6.

[37] E. A. Lee et al, "Overview of the Ptolemy Project," UCB/ERL M01/11, University of California, Berkeley, USA, Tech. Rep., 2001.

[38] R. Ernst, D. Ziegenbein, K. Richter, L. Thiele, and J. Teich, "Hardware/Software Co-Design of Embedded Systems The SPI Workbench," in *Proceedings of the International Workshop on VLSI, Orlando, Florida, USA*, 1999.

[39] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Embedded Control Systems Development with Giotto," in *Proceedings of the LCTES*, 2001.

[40] E. Closse, M. Poize, J. Pulou, J. Sifakis, P. Venier, D. Weil, and S. Yovine, "TAXYS = ESTEREL + KRONOS. A Tool for the Development and Verification of Real-Time Embedded Systems," in *Proceedings of the CAV'01, LNCS, Springer-Verlag*, vol. 2102, 2001, pp. 391–395.

[41] G. Berry and G. Gonthier, "The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation," *Science of Computer Programming*, vol. 19, no. 2, pp. 87–152, 1992.

[42] D. Weil, V. Bertin, E. Closse, M. Poize, P. Vernier, and J. Pulou, "Efficient Compilation of Esterel for Real-Time Embedded Systems," in *Proceedings of the CASES'2000*, 2000.

[43] G. Bhat, R. Cleaveland, and G. Luttgen, "A Practical Approach to Implementing Real-Time Semantics," *ASE*, vol. 7, pp. 127–155, 1999.

[44] R. Milner, "Communication and Concurrency," 1989.

[45] A. N. Fredette and R. Cleaveland, "RTSL: A Language for Real-Time Schedulability Analysis," in *Proceedings of the RTSS'93*. Computer Society Press, 1993, pp. 274–283.

[46] I. Lee, P. Brmond-Grgoire, and R. Gerber, "A Process Algebraic Approach to the Specification and Analysis of Resource-bound Real-Time Systems," in *Proceedings of the IEEE, Special Issue on Real-Time Systems*, vol. 1, 1994.

[47] P. Brmond-Grgoire and I. Lee, "A Process Algebra of Communicating Shared Resources with Dense Time and Priorities," *Theoretical Computer Science*, vol. 189, 1997.

[48] O. Sokolsky, I. Lee, and H. Ben-Abdallah, "Specification and Analysis of Real-Time Systems with PARAGON," pp. 211–234, 1999.

[49] H. Ben-Abdallah, J.-Y. Choi, D. Clarke, Y.-S. Kim, I. Lee, and H. liang Xie, "A Process Algebraic Approach to the Schedulability Analysis of Real-Time Systems," *Real-Time Systems*, vol. 15, no. 3, pp. 189–219, 1998.

[50] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority-Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, vol. 39, pp. 1175–1185, 1990.

[51] T. Janowski and M. Joseph, "Dynamic Scheduling and Fault-Tolerance: Specification and Verification," *Real-Time Systems*, vol. 20, no. 1, pp. 51–81, 2001.

[52] Christer Norström Ericsson and Anders Wall and Wang Yi, "Timed Automata as Task Models for Event-Driven Systems," in *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications*. Washington, DC, USA: IEEE Computer Society, 1999, p. 182.

[53] V. A. Braberman, "Modeling and Checking Real-Time System Designs," Ph.D. Thesis, Departmento de Computacion, Universidad de Buenos Aires, Argentina, 2000.

[54] S. Vestal, "MetaH Support for Real-Time Multi-Processor Avionics," in *5th IEEE Workshop on Parallel and Distributed Real-Time Systems*, 1997, pp. 132–137.

[55] S. Vestal, "Fixed-Priority Sensitivity Analysis for Linear Compute Time Models," *IEEE Transactions on Software Engineering*, vol. 20, no. 4, pp. 308–317, 1994.

[56] Pam Binns, "Incremental Rate Monotonic Scheduling for Improved Control System Performance," in *Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium (RTAS)*. Washington, DC, USA: IEEE Computer Society, 1997, p. 80.

[57] C. J. Fidge, P. Kearney, and M. Utting, "Interactively Verifying a Simple Real-time Scheduler," in *Proceedings of the 7th International Conference on Computer Aided Verification*. London, UK: Springer-Verlag, 1995, pp. 395–408.

[58] B. Dutertre, "Formal Analysis of the Priority Ceiling Protocol," in *Proceedings of the Real-Time Systems Symposium*, Orlando, Florida, USA, 2000.

[59] B. Dutertre and V. Stavridou, "Formal Analysis for Realtime Scheduling," in *Proceedings of the 19th AIAA/IEEE Digital Avionics Systems Conference*, Philadelphia, USA, 2000.

[60] Z. Liu and M. Joseph, "Specification and Verification of Fault-Tolerance, Timing and Scheduling," pp. 46–89, 1999.

[61] ——, "Verification, Refinement and Scheduling of Real-time Programs," pp. 119–152, 2001.

[62] L. Lamport, "The Temporal Logic of Actions," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 3, pp. 872–923, May 1994.

[63] P. Wielage and K. Goossens, "Networks on Silicon: Blessing or Nightmare?" in *Proceedings of the Euromicro Symposium on Digital Systems Design*. Washington, DC, USA: IEEE Computer Society, 2002, p. 196.

[64] Narayanan Swaminathan and Rabi Mahapatra, "Communication Architecture Synthesis of Packet-Switched Network-on-Chip," TR-CS-2002-08-0, Tech. Rep.

[65] Praveen Bhojwani and Rabi N. Mahapatra, "Interfacing Cores with On-chip Packet-Switched Networks," in *Proceedings of the 16th International Conference on VLSI Design*, January 2003, pp. 382–387.

[66] Luca Benini and Giovanni De Micheli, "Powering Networks on Chips," in *Proceedings of the ISSS*, October 2001, pp. 33–38.

[67] Jingcao Hu and Radu Marculescu, "Energy-Aware Mapping for Tile-based NoC Architectures under Performance Constraints," in *Proceedings of the Asia & South Pacific Design Automation Conference (ASP-DAC)*, January 2003.

[68] B. Vermeulen, J. Dielissen, K. Goossens, and K. Ciordas, "Bringing Communication Networks On Chip: Test and Verification Implications," *IEEE Communications Magazine*, vol. 41, no. 9, pp. 74–81, September 2003.

[69] Mohsen Nahvi and Andre Ivanov, "A Packet Switching Communication-based Test Access Mechanism for System Chips," in *Proceedings of the 6th IEEE European Test Workshop*, 2001, p. 81.

[70] T. Dumitras, S. Kerner, and R. Marculescu, "Towards On-Chip Fault-Tolerant Communication," in *Proceedings of the Asia & South Pacific Design Automation Conference (ASP-DAC)*, January 2003.

[71] "OPNET," http://www.opnet.com.

[72] Luca Benini and Giovanni De Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, January 2002.

[73] William Dally and Brian Towles, "Route Packets not Wires: On-Chip Interconnection Networks," in *Proceedings of the DAC*, 2001.

[74] M. Gasteiner and M. Glessner, "Bus-based Communication Synthesis on System Level," *ACM Transactions on Design Automation of Electronic Systems*, vol. 19, pp. 1–11, January 1999.

[75] Peter Voigt Knudsen and Jan Madsen, "Integrating Communication Protocol Selection with Partitioning in Hardware/Software Codesign," in *Proceedings of the International Symposium on System Synthesis*, 1998.

[76] A. Baghdadi, D. Lyonnard, N.-E. Zergainoh, and A. Jerraya, "An Efficient Architecture Model for Systematic Design of Application-Specific Multiprocessor SoC's," in *Proceedings of the International Conference on Design and Test in Europe (DATE)*, 2001.

[77] K. Hines and G. Borriello, "Dynamic Communication Models in Embedded System Cosimulation," in *Proceedings of the Design Automation Conference (DAC)*, 1997.

[78] K. Lahiri, A. Raghunathan, and S. Dey, "Performance Analysis of Systems with Multi-Channel Communication Architectures," in *Proceedings of the International Conference on VLSI Design*, 2000, pp. 530–537.

[79] P. G. Paulin, C. Pilkington, and E. Bensoudane, "StepNP: A System-Level Exploration Platform for Network Processors," *IEEE Design and Test of Computers*, vol. 19, no. 6, pp. 17–26, December 2002.

[80] S. Goosens, T. Kogel, M. Doerper, A. Wieferink, R. Laupers, G. Ascheid, and H. Meyr, "A Modular Simulation Framework for Architectural Exploration of On-Chip Networks."

[81] J. Madsen, S. Mahadevan, and K. Virk, "Network-on-Chip Modeling for System-Level Multiprocessor Simulation," in *Proceedings of the 24th IEEE Real-Time Systems Symposium*, December 2003, pp. 265–274.

[82] A. Savvides, S. Park, and M. B. Srivastava, "On Modeling Networks of Wireless Microsensors," *ACM*, 2001.

[83] Alice Wang and Rex Min and Masayuki Miyazaki and Amit Sinha and Anantha Chandrakasan, *Low-Power Sensor Networks, The Applications of Programmable DSP's in Mobile Communications*. John-Wiley & Sons, 2002.

[84] J. Feng, F. Koushanfar, and M. Potkonjak, "System Architectures for Sensor Networks - Issues, Alternatives and Directions," in *Proceedings of International Conference on Computer Design*, September 2002.

[85] Domenico Talia, "Parallel Computation Still not Ready for the Mainstream," *ACM*, vol. 40, no. 7, July 1997.

[86] N. K. Tibrewala, J. M. Paul, and D. E. Thomas, "Modeling and Evaluation of Hardware/Software Designs," in *Proceedings of the 9th International Symposium on Hardware/Software Codesign*, 2001, pp. 11–16.

[87] Edward A. Lee, "What's Ahead for Embedded Software?" *IEEE Computer*, pp. 18–26, September 2000.

Chapter 3

A System-level Multiprocessor System-on-Chip Modeling Framework

Jan Madsen, Kashif Virk and Mercury Jair Gonzalez. A System-level Multiprocessor System-on-Chip Modeling Framework. *Proceedings of the IEEE International Symposium on System-on-Chip, 2003.* (SoC'03), November 2003. Pages 147-150. Published.

A System-level Multiprocessor System-on-Chip Modeling Framework

Jan Madsen Kashif Virk Mercury Jair Gonzalez

Computer Science & Engineering Section
Department of Informatics & Mathematical Modeling
Technical University of Denmark, Lyngby 2800, Denmark
email: {virk, jan}@imm.dtu.dk

***Abstract*—In this paper, we present a SystemC-based framework to study the effects of running multi-threaded application software on a multiprocessor platform under the control of one or more abstract real-time operating systems (RTOSs). We propose a modelling framework consisting of basic RTOS service models; scheduling, synchronization, and resource allocation, and a generic task model that is able to model periodic and aperiodic tasks as well as task properties such as varying execution times, offsets, deadlines, and data dependencies. A given multiprocessor system is formed by the composition of RTOS service models and the allocation of tasks (the application software) onto RTOSs. We demonstrate the potential of our approach by simulating and analyzing a small multiprocessor system.**

I. Introduction

As embedded systems become more and more complex, todays applications demand a considerable computational power from their platforms. To match these requirements, it becomes necessary to utilize the parallel and distributed systems technology. There is a growing trend towards the implementation of heterogeneous architectures consisting of several programmable, as well as, dedicated processors on a single chip. As an increasing portion of applications are implemented in software which, in turn, is growing larger and more complex, dedicated operating systems will have to be introduced as an interface layer between the application software and the hardware platform [2]. Global analysis of such heterogeneous systems is a big challenge. Typically, two aspects are of interest when considering global analysis: the system functionality and the timing and resource sharing, in particular. Our aim is to study embedded applications executing on a multiprocessor platform running a number of, possibly, different RTOSs. As many embedded applications are reactive in nature and have real-time requirements, it is often not possible to analyze them statically at compile-time. Furthermore, for single-chip solutions, we may need to use non-standard RTOSs in order to limit the code size and, hence, memory requirements, or to introduce special features interacting with the dedicated hardware, such as power management. When implementing an RTOS, we may wish to experiment with different scheduling strategies in order to tailor the RTOS to the application. For a multiprocessor platform, we may wish to study the system-level effects of selecting a particular RTOS implementation on one of the processors. To study these effects at the system-level, before any implementation has been done, we need a system-level model which is able to capture the behaviour of running a number of RTOSs on a multiprocessor platform. In this paper, we propose a framework to model abstract application software (modelled as a set of task graphs) executing on a multiprocessor platform under the supervision of abstract RTOSs. The framework is based on SystemC 2.0 [6].

II. Related Work

Validation of multiprocessor RTOS's is a complicated process which is often solved in an ad-hoc manner due to the lack of uniform methodologies and tools that cover all the aspects pertaining to the modelling of modern heterogeneous systems. This has been discussed in [2], [3], [5], [10] and several approaches to develop such a methodology have been devised. Sifakis [9] presents a methodology based on composition to model real-time systems although nothing is mentioned about the challenges implied by the modelling of real-time systems implemented on multiprocessor platforms. The approach followed by METAH [12] and VEST [11], is based on the functional description of multiprocessor real-time systems giving modelling capabilities and automatic generation of different components including the operating system. However, the focus is at a lower abstraction level than the one we propose. In [2], a high-level performance model for multi-threaded, multiprocessor systems is presented. This approach is based on modelling the layer of schedulers in an abstract manner, which resembles the aim of our approach. Others have focused on providing RTOS modelling on top of existing System Level Design Languages (SLDL), either for open languages such as SpecC [5] and the RTOS library of SystemC 3.0 [1], or for proprietary languages such as SOCOS [4] of OCAPI and TAXYS [10]. These approaches offer functional models of the RTOS enabling its emulation, on top of which functional models of software applications can be implemented.

III. Framework

At the system levelthe application software may be modelled as a set of tasks, $\tau_i \in T$, which have to be executed on a number of programmable processors under the control of one or more RTOS(s). Our system model is designed following the principle of composition as described in [9] and consists

of three types of basic components: tasks, RTOS services, and links, where the links provide communication between other system components. The RTOS services are decomposed into independent modules that model different basic RTOS services: A scheduler models a real-time scheduling algorithm. A synchronizer models the dependencies among tasks and, hence, both intra- and inter-processor communications. And an allocator models the mechanism of resource sharing among tasks.

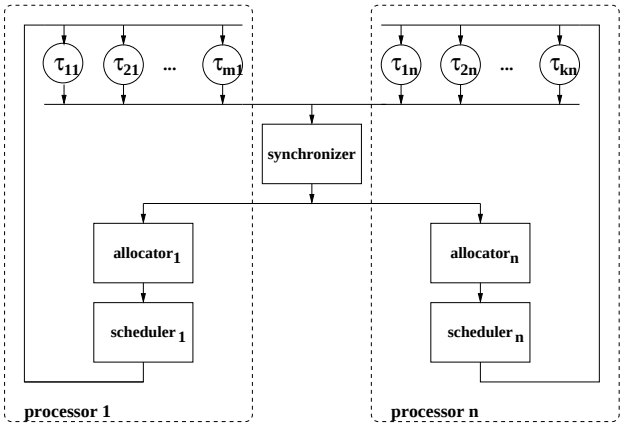


Fig. 1. *Architecture of the System Model*

In this paper, we assume that each processor can run just one scheduling algorithm. In a multiprocessor platform, we would have a number of schedulers representing the same number of processors, while synchronization and allocation may be represented by a single instance of each. Tasks can send the messages: `ready` and `finished`, to the scheduler which, in turn, can send three commands to the tasks: `run`, `preempt`, and `resume`. In between the schedulers and the tasks, we have the synchronizer and the allocator acting as logical message filters as shown in Figure 1. As a way to maintain composition, each component handles its relevant data, independently of the other. For example, a task determines when it is ready to run and when it has finished. In this way, the scheduler behaves in a reactive manner; scheduling tasks according to the data received from them. Thus, we can add as many tasks and schedulers as we desire. The same is the case with the synchronizer and the allocator models. They hold the information regarding their services, i.e., which tasks depend on which other or, for the case of the allocator, what resources are needed by a given task. We use a global clock connected to all tasks (not shown in Figure 1) to measure time in terms of clock cycles, i.e., use an abstract time unit. This allows us to identify the moment at which a task is ready to be executed or when a task has finished its execution.

A. Task Model

At the system level, we are not interested in how a task, τ_i, is implemented, i.e., its exact functionality, but we need information regarding the execution of the task, such as the *WCET*, *BCET*, *context switching overhead*, *period* (T_i), *deadline* (d_i), and *offset* (o_i), in order to characterize execution of the task. The dependencies among tasks are handled in another part of our model which will be explained later. The behaviour of a task is modelled as a finite state machine (FSM) with four states: idle, ready, running, and preempted. See Figure 2. We assume that all the tasks start in the idle state with a certain offset[1] that can have any value including zero; in which case, the task goes immediately to the ready state, waiting for the RTOS to issue a `run` command.

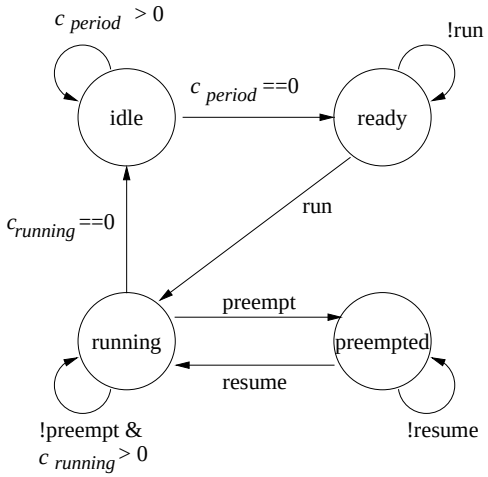


Fig. 2. *The Task Model*

The task stays in the ready state until it receives a run command from the scheduler. It then goes to the running state in which it counts the number, crunning, of cycles. When entering the running state, crunning is initialized to the value of the task execution time c_i[2]. When $c_{running} == 0$, the task has finished its computation. It then issues a `finished` message to the scheduler and goes to the idle state. In all states, c_{period} is decremented each cycle. After reaching the idle state, the task stays there until $c_{period} == 0$ indicating the start of a new period by making a transition to the ready state and setting $c_{period} = T_i$. At any time during the running state, the task may be preempted by the scheduler, i.e., the scheduler sends a preempt command to the task. When preempted, the task goes into the preempted state where it waits for the resume command from the scheduler. During the preempted state, only the value of c_{period} is updated.

B. Scheduler Model

From a system point of view, the major task of the RTOS is to determine the execution order of the tasks, i.e., to perform task scheduling. The scheduler maintains a list of tasks ready to be executed. In our model, the list is a priority queue where each task is given a priority according to the scheduling policy of the scheduler. For example, for the rate-monotonic (RM) scheduling, the task priority is based on the period of the task while for the deadline-monotonic (DM) and the earliest-deadline-first (EDF) scheduling, the task priority is based on the tasks deadline. In the following, we use $p(\tau_i)$ to denote the

[1]The offset is the time from the start of the system to the first time a given task gets ready. This is, sometimes, referred to as the phase

[2]The execution time, c_i, is calculated as a random number between the $BCET$ and the $WCET$ of the particular task.

priority of task τ_i. The scheduler is modelled as an event-based process that runs whenever a message (`ready` or `finished`) is received from a task. In the case of a finished message received from a task τ_i, the scheduler selects, from the list of ready tasks, the one with the highest priority, τ_j, and issues the command `run` to τ_j. In case of an empty list and no running task, the scheduler just waits to receive a `ready` message. As soon as it receives this message, it issues a `run` command to the ready task τ_i. If the list is empty, but a task, τ_k, is currently running, then,

1) if $p(\tau_k) > p(\tau_i)$ then τ_i enters the list of ready tasks.
2) if $p(\tau_k) < p(\tau_i)$ then τ_k is preempted by issuing a `preempt` command to τ_k and placing τ_i in the ready list. Then the scheduler issues a `run` command to τ_i.

If the ready list is not empty, τ_i is only executed if it has the highest priority, otherwise, the task, τ_j, with the highest priority is selected.

The scheduler is designed to attend to several messages in zero simulation time. When two or more different tasks send a ready message simultaneously to the scheduler in the same simulation cycle, the scheduler will choose the task with the highest priority to run and enqueue the others. This is handled very elegant by connecting Master ports to Slave ports using the SystemC Master-Slave library[3]. The Master-Slave library ensures that tasks are actually served sequentially during the simulation cycle, although, the order is non-deterministic.

C. Synchronization Model

Another of the basic services provided by an RTOS is synchronization among the cooperative tasks that are running in parallel. For example, if τ_i needs the data computed by τ_j, then τ_i has to wait till the completion of τ_j in order to execute. As we have designed our framework to support multiprocessor system environments, the synchronizer handles intra- and inter-processor dependencies as well as multi-rate systems[4]. Task dependency can be of various types, but at the system level, we do not care about the nature of a dependency. We can formulate an abstraction and assert that task, τ_j, is eligible to be released just after the task τ_i has finished its execution. The synchronizer can be seen as a message filter between the tasks and the schedulers letting other schedulers know when a task is really ready, i.e., when its dependency constraints have been resolved. The `finished` message will always pass but the `ready` message will pass only when the dependency constraints have been resolved. Every time a task issues a message, the synchronizer will receive it. Its reaction will depend on the implemented synchronization protocol. The basic synchronizer in our model works in the following way: When it receives a `finished` message, it looks into the dependency database to see if the issuing task has dependencies. If so, it checks the waiting list if these dependencies are already waiting. If they are waiting, then they are released to the scheduler. If not, information about the issuing task is stored in the finished list. If the received message is ready, the synchronizer looks into the dependencies database to see if the task has dependencies, it then checks if its dependencies are already in the finished list. If they are, the finished task is removed from the list and the ready message is passed to the scheduler, otherwise, information about the issuing task is stored in the waiting list.

[3]www.systemc.org

[4]The period of the producer is different from that of the consumer

D. Resource Allocation Model

The Resource Allocator uses the Priority Inheritance Protocol [8] for allocating the resources requested by the tasks. The Priority-Inheritance Protocol ensures that, in the absence of Deadlocks, no task is ever blocked for an indefinitely long time because an uncontrolled Priority Inversion cannot occur. When a task requires a resource, it sends a `request` message to the allocator which either issues a `grant` message to the scheduler if the requested resource is available or a `refuse` message if it is not. In both the cases, the priority of the task requesting the resource is updated in accordance with the Priority Inheritance Protocol and is notified to the scheduler by an `updatePriority` message. In a similar way, when a task has occupied a resource for its designated duration, it sends a `release` message to the allocator which updates its resource database and issues an `updatePriority` message to the scheduler as demanded by the Priority Inheritance Protocol.

IV. Results

In this section, we will illustrate the capabilities of our framework by analyzing a small multiprocessor example.

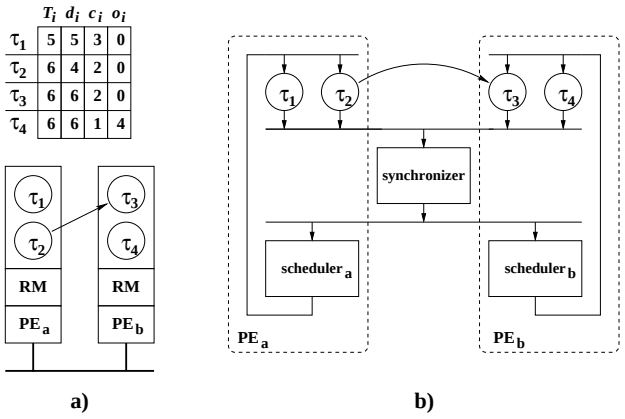


Fig. 3. *a) Task characterization and allocation for the multiprocessor example; b) Modeling the example*

Figure 3a shows a system with two processors, running two such tasks each. A single dependency exists between tasks τ_2 and τ_3. As seen in the figure, it is an inter-processor dependency. The figure also shows the characterization of each of the tasks. Figure 3b shows how we model the example using our abstract RTOS model. Notice that, in this example, we have omitted the resource allocator. In the first approach, we will use rate-monotonic (RM) scheduling as the scheduling policy for the RTOS on both processors. Figure 4 shows the behaviour of the system in terms of a waveform indicating the state changes over time for each task in the system. As τ_1 has a shorter period than τ_2, it has, according to the RM policy, a higher priority and, hence, starts executing at time 0. After

3 time units, τ_1 has completed its execution well ahead of its deadline. τ_2 can then run until it completes after 2 time units at time 5. Due to the dependency between τ_2 and τ_3, τ_3 has to wait until time 5. As τ_3 has a period of 6 time units and a delay of 2 time units, it misses its deadline at time 6. If we change the scheduling policy of the RTOS on processor PEa to the earliest-deadline-first (EDF), τ_1 and τ_2 are now executed in a different order, which allows τ_2 to deliver its data to τ_3 in time for τ_3 to meet its deadline.

V. Conclusions

We have presented a modelling framework based on SystemC which supports the modelling of multiprocessorbased RTOSs. The aim of the framework is to provide the system designer with a user-friendly and efficient modelling and simulation environment in which he/she can experiment with different RTOS policies and study the consequences of local decisions on the global system behaviour. So far, our test cases have been aimed at providing the proof-of-concept as is the case for the example presented in the previous section. However, we are, currently, working on several large real-life examples including a GSM encoder/decoder [7] containing 87 tasks.

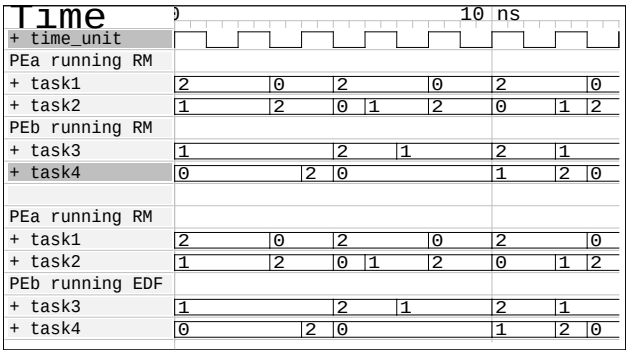


Fig. 4. *Schedule of the Example. In the top figure, both processors are running RM scheduling, whereas the scheduler on processor PEb is changed to EDF in the lower figure. Symbols: 0=idle, 1=ready, 2=running and 3=preempted*

References

[1] "Modeling Software with SystemC," European SystemC Users' Group Meeting, October 2002.

[2] A. S. Cassidy and J. M. Paul and D. E. Thomas, "Layered, Multi-Threaded, High-Level Performance Design," in *Design Automation and Test in Europe, DATE*, March 2003, pp. 954–959.

[3] W. Cesario, L. Gauthier, D. Lyonnard, G. Nicolescu, and A. Jerraya, "An XML-based Meta-Model for the Design of Multiprocessor Embedded Systems," in *VHDL International User's Forum*, October 2000.

[4] D. Desment, D. Verkest, and H. D. Man, "Operating System-based Software Generation for Systems-on-Chip," in *Design Automation Conference, DAC*, June 2000.

[5] A. Gerstlauer, H. Yu, and D. Gajski, "RTOS Modeling for System-Level Design," in *Design Automation and Test in Europe, DATE*, March 2003, pp. 132–137.

[6] T. Grotker, G. M. S. Liao, and S. Swan, *System Design with SystemC*. New York: Kluwer Academic Publishers, 2002.

[7] M. Schmitz, B. Al-Hashimi, and P. Eles, "A Co-Design Methodology for Energy-Efficient, Multi-Mode Embedded Systems with the consideration of Mode Execution Probabilities," in *Design Automation and Test in Europe, DATE*, March 2003, pp. 960–965.

[8] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority-Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, vol. 39, pp. 1175–1185, 1990.

[9] J. Sifakis, "Modeling Real-Time Systems - Challenges and Work Directions," in *EMSOFT, Lecture Notes in Computer Science 2211*, October 2001.

[10] J. Sifakis, S. Tripakis, and S. Yovine, "Building Models of Real-Time Systems from Application Software," *IEEE Special Issue on Modelling and Design of Embedded Systems*, pp. 100–111, January 2003.

[11] A. Stankovic, H. Wang, M. Humphrey, R. Zhu, R. Poornalingam, and C. Lu, "VEST: Virginia Embedded Systems Toolkit," in *Real-Time Embedded Systems Workshop*, December 2001, pp. 132–137.

[12] S. Vestal, "MetaH Support for Real-Time Multi-Processor Avionics," in *5th IEEE Workshop on Parallel and Distributed Real-Time Systems*, 1997, pp. 132–137.

Chapter 4

Network-on-Chip Modelling for System-Level Multiprocessor Simulation

Jan Madsen, Shankar Mahadevan, Kashif Virk and Mercury Gonzalez. Network-on-Chip Modelling for System-Level Multiprocessor Simulation. *Proceedings of the IEEE Real-Time Systems Symposium* (RTSS 2003), December 2003. Pages: 265-274. Published.

Network-on-Chip Modeling for System-Level Multiprocessor Simulation

Jan Madsen Shankar Mahadevan Kashif Virk Mercury Gonzalez

Computer Science & Engineering Section
Department of Informatics & Mathematical Modeling
Technical University of Denmark, Lyngby 2800, Denmark
email: {jan, sm, virk}@imm.dtu.dk

***Abstract*—With the increasing number of transistors available on a single chip, the System-on-Chip (SoC) paradigm has evolved to exploit its full potential. As many processors can be accommodated on a single chip, this paradigm has forced a communication-centric, as opposed to a computation-centric, design view. Thus, the choice, management and modeling of the SoC interconnect is essential for an accurate evaluation and optimization of the global performance of a system. Recently, the notion of Network-on-Chip (NoC) has been introduced as a way to extend the classical bus-based interconnection, which is still the dominant interconnect structure for SoC's, into a dedicated, segmented and, possibly, packet-switched network fabric [2]. In this paper, we present a NoC model which, together with a multiprocessor real-time operating system (RTOS) model, allows us to model and analyze the behavior of a complex system that has a real-time application running on a multiprocessor platform. We demonstrate the potential of our model by simulating and analyzing a small multiprocessor system connected through different NoC topologies, and discus how the simulation model may be used during the design-space exploration phase.**

I. Introduction

With the growing complexity of embedded systems and the capacity of modern silicon technology, there is a trend towards heterogeneous architectures consisting of several programmable and dedicated processors, implemented on a single chip, known as a System-on-Chip (SoC). As an increasing portion of applications is implemented in software which, in turn, is growing larger and more complex, dedicated operating systems will have to be introduced as an interface layer between the application software and the hardware platform [5]. On the other hand, the hardware platform will either be developed as a part of the design process or configured from an existing reconfigurable platform, which allows for the implementation of parts of an application as dedicated processors (ASIC's).

Modern silicon technologies, with minimum device geometries in the nanometer range ($<$100nm), have made it possible to integrate hundreds of processors on a single chip. In these deep submicron technologies, the on-chip interconnection fabric is a major source of delay and power consumption which is challenging the on-chip communication infrastructure and forcing a change from device-centric to interconnect-centric design methodologies. Traditionally, on-chip communication has either been conducted via dedicated point-to-point links or by shared media like a bus. Neither is very suitable for generalized communication handling in large systems [13]. A promising solution is to have a dedicated, segmented, and, possibly, packet-switched network fabric on the chip, a Network-on-Chip (NoC) [2].

Hence, when mapping an application onto its target platform, hardware/software codesign aspects [18] have to be taken into account. These include mapping of tasks onto software, hardware, or a combination of both, as well as task dependencies on the communication infrastructure. In order to do so, accurate modeling of the systems and all the interrelationships among the diverse processors, software processes and physical interfaces and interconnections, is needed. One of the the primary goals of system-level modeling is to formulate a model within which a broad class of designs can be developed and explored. To support the designers of single-chip based embedded systems, which includes multiprocessor platforms running dedicated real-time operating systems (RTOS's) as well as the effects of on-chip interconnect network, a system-level modeling/simulation environment is required to support an analysis of the:

- consequences of different mappings of tasks to processors (software or hardware),
- network performance under different traffic and load conditions,
- effects of different RTOS selections, including various scheduling, synchronization and resource allocation policies.

In this paper, we present a modeling environment based on SystemC [22] which can provide the SoC designers a software-like, system-level abstraction of the platform as well as supporting the three requirements mentioned above for system-level design-space exploration.

Most of the future embedded applications are likely to be real-time applications that will run on multiprocessor SoC's which are, essentially, distributed computing systems. In a multiprocessor or a distributed system, the processing elements can be connected through shared memory, dedicated communication links or a communication network. Instead of dealing with each specific application and system architecture, we deal with generalized abstract tasks, processing elements, and communication infrastructures. This not only broadens the applicability of our modeling framework, but also leads to a

better understanding of the problem at hand.

We extend our previous work [9], [16] on the modeling of a multi-threaded application, running on a multiprocessor platform under the control of one or more abstract RTOS's, with a model of an on-chip network which can provide provisions for run-time inspection and observation of the on-chip communication. Using this system-level design approach, implementations of the most promising network alternatives can be prototyped and characterized in terms of performance and overhead. Taking communication into account during hardware/software mapping is essential in order to obtain optimized solutions as emphasized in [14].

The paper is organized as follows: Section II describes current trends and related work in the field of communication network modeling for multiprocessor environments. In Section III, we provide a brief overview of our previously proposed RTOS model and discuss its extension to include the NoC model. Section IV presents our main ideas on NoC modeling. It provides the methodology for developing a network model for usage at the system-level. This model seamlessly handles the allocation and scheduling of communication events within the NoC as driven by the requirements from the tasks running on the PE's in a SoC. A SystemC implementation of a torus network is also discussed. The results of our implementation and simulation of the model are given in Section V. Further, in Section VI, we extend this discussion to the effects of select design-space exploration choices on global system performance. Section VII, finally, provides conclusions and the future direction of our work.

II. Related Work

One of the essential elements of making a transition from ad-hoc system-on-chip (SoC) designs to a disciplined SoC design approach is taking a rigorous, though flexible, approach towards the design of on-chip communication networks that interconnect IP blocks of all variety, including the processing elements (PE's). A network-on-chip (NoC) approach, driven by a consistent design methodology, is bound to lead to dramatic changes in how SoC's will be designed in the future. The partitioning and mapping of tasks onto complex architectures (homogeneous or heterogeneous) is a well-known hardware/software codesign problem [18]. [8], [12], [16]–[19] further explain allocation, scheduling and synchronization in RTOS's. But the notion of the on-chip communication medium has been quite primitive. It has, generally, been viewed as an overhead where no other useful work can be accomplished. Thus, it is assumed to occur instantaneously or it is given a token fixed overhead time. This approach is suboptimal and error-prone requiring further iteration before design closure. [12] and [14] clearly show the importance of evaluating the communication media and how the choice of a communication architecture clearly impacts the overall architecture of a SoC. In [1], a communication model for codesign has been described, but it is limited and cannot account for specific NoC features for design-space exploration at the system level.

There, already, exists plenty of research literature on the communication modeling for multiprocessors with different interconnection topologies to characterize their communication performance, for example [3]. Moreover, in [2] and [23], the concept of on-chip, packet-switched micro-networks has been introduced that borrows ideas from the layered design methodology for data networks. In [15] the layered, packet-switched NoC design concepts have been applied to a 2-D Mesh Network Topology whereas in [10], similar concepts have been applied to a Butterfly Fat Tree Topology. While there are several mature methodologies for modeling and evaluating the processing element architectures, there is relatively little research done to port the on-chip communication to system-level. In [24], attempts have been made to fill this gap by proposing a NoC modeling methodology based upon the ideas borrowed from the object-oriented design domain and implementing those ideas using an existing CAD framework – Ptolemy II. However, the authors have conjectured about the performance gains achievable by the porting of their proposed modeling framework to SystemC. In [21], a theoretical framework for modeling real-time applications running on multiprocessor systems has been developed that models the inter-processing element communication with a link processor. But such attempts are quite ad-hoc and no generalized approach has, so far, been reported to our knowledge.

In our proposed abstract system modeling framework, an embedded, real-time application is represented as a collection of multiple, concurrent execution threads that are modeled as a set of dependent tasks under certain precedence and resource constraints. Such tasks, in turn, are modeled as a chain of sub-tasks executing on, possibly, different processing elements. Based on the abstract system model, three distinct, but closely-related problems are identified, namely, execution synchronization, resource allocation and priority assignment/scheduling. The inter-processing element communication is modeled by modeling a communication network as a communication processor and the message transmission through the network as a communication task running (concurrently) on the communication processor. Using this approach, we have demonstrated that our, previously proposed [9], [16], abstract RTOS model can be extended to include an abstract NoC processor that can effectively model the system-level effects of any NoC architecture.

III. Abstract RTOS Modeling

As discussed earlier, at the system level, the application software may be modeled as a set of tasks which have to be executed on a number of processing elements (PE's) under the control of one or more RTOS(s). For details on the model and how it is implemented in SystemC (including the use of the Master-Slave library), we refer to [9] and [16].

Briefly, our system model is designed following the principle of composition, as described in [20], and consists of three types of basic components: tasks, RTOS services, and links, where the links provide communication between other system components. We have used SystemC 2.0 as the implementation language of our model. Although, any language could have been used, the choice of SystemC is mainly due to the fact that it is an extension of the C++ programming language and has a built-in simulation kernel that supports concurrency. In addition, it supports the design process from system-level

down to both hardware and software implementations. The SystemC Master-Slave library provides a very elegant way of handling concurrent messages sent by the tasks to the RTOS services. This allows each RTOS service to deal with a single message at a time independently of the other. Figure 1 shows the Abstract RTOS Model and Figure 2 presents the overall system model, including the NoC model which will be described in the next section. In this section, we focus on the RTOS modeling which corresponds to the PE's. The RTOS services are composed from independent modules that model different basic RTOS services. A scheduler models a real-time scheduling algorithm. A synchronizer models the dependencies among tasks and, hence, both intra- and inter-processing element communications. An allocator models the mechanism of resource sharing among tasks.

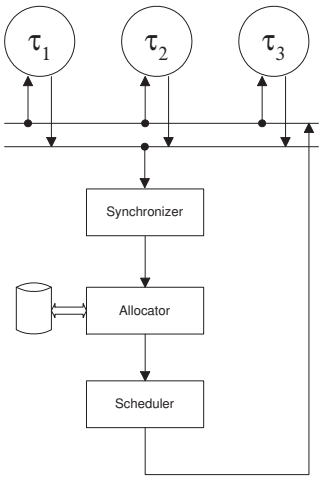


Fig. 1. *Abstract RTOS model.*

The model is designed such that any of the RTOS services can be changed in a simple and straight forward manner. Tasks are considered to be abstract representations of the application and are characterized by a set of parameters, such as the worst- and the best-case execution time, context switching overhead, deadline, period (if it is a periodic task), offset, resource requirements, and precedence relations. A task is modeled as a finite state machine (FSM) which can send the messages: `ready` and `finished`, to the scheduler which, in turn, can send one of the three commands to the tasks: `run`, `preempt`, and `resume`. In between the schedulers and the tasks, we have the synchronizer and the allocator acting as "logical command filters". As a way to maintain composition, each module handles its relevant data independently of the other. For example, a task determines when it is ready to run and when it has finished. In this way, the scheduler behaves in a reactive manner; scheduling tasks according to the data received from them. Thus, we can add as many tasks and schedulers as we desire. The same is the case with the synchronizer and the allocator models. They hold the information regarding their services, i.e., which tasks depend on each other or, for the case of the allocator, what resources are needed by a given task.

IV. NoC Modeling

Architecturally, a network is characterized by its *topology* and the *protocol* running on it. The topology concerns the geometry of the communication links on the chip while the protocol governs the usage of these links. Many combinations of topology and protocol exist for the efficient communication of one or more predominant traffic patterns. The *performance* of a network is measured in quantitative terms such as latency, bandwidth, power consumption and area usage, and in qualitative terms such as network reconfigurability (dynamic or static), quality of service (QoS), etc. Predictability of performance is necessary for NoC designers to take early decisions based on the NoC performance before actual implementation. Numerous studies have been done for deadlock, livelock, congestion-avoidance, error-correction, network setup/tear-down, etc. to provide a certain predictable network behavior [7]. Even lower-level engineering techniques like low-swing drivers, signal encoding etc., have been proposed to overcome network communication uncertainties [4], [6], [11]. Many of these aspects are custom-tuned to fit the requirements of the underlying application.

Throughout this paper, we use *network latency* as a primary factor for grading the performance of a network. The network latency is defined as the time taken to move data from a source PE to a destination PE. It includes the message processing overhead at the PE's, link delays and the data processing delays at the intermediate nodes [14]. It is a function of the topology (which determines the number of nodes and links) and the protocol (which defines the processing requirements for routing and flow-control).

The *state* of a network at any instant is given by the number of actively transmitting PE's and the messages within its nodes and links. The state of a network dictates which resources of the network are currently in use and which ones can be available for future use. This provides a measure of the *network services* available to the system, which affect its performance. We define network services as the system-level characterization of network resource allocation and scheduling. For a given topology-protocol combination, changes in network services, change the resources available for a given communication event, thus, affecting its latency.

For the purpose of forming a system-level NoC simulation model, unlike a network simulator, we have abstracted away all the above-mentioned low-level network details except the most essential ones (e.g., topology, latency, etc.). We treat the on-chip communication network as a *communication processor* to reflect the servicing demands. A communication event within this network is modeled as a *message task*, τ_m, executing on the communication processor. When one PE wants to communicate with another PE, a τ_m is fired on the communication processor. Each τ_m represents communication only between two fixed set of predetermined PE's. Since a NoC supports concurrent communication, τ_m's need to be synchronized, allocated resources and scheduled accordingly. This is a property of the underlying NoC implementation, where the NoC allocator reflects the topology and the NoC scheduler reflects the protocol. A resource database, which is unique to each NoC implementation, contains information

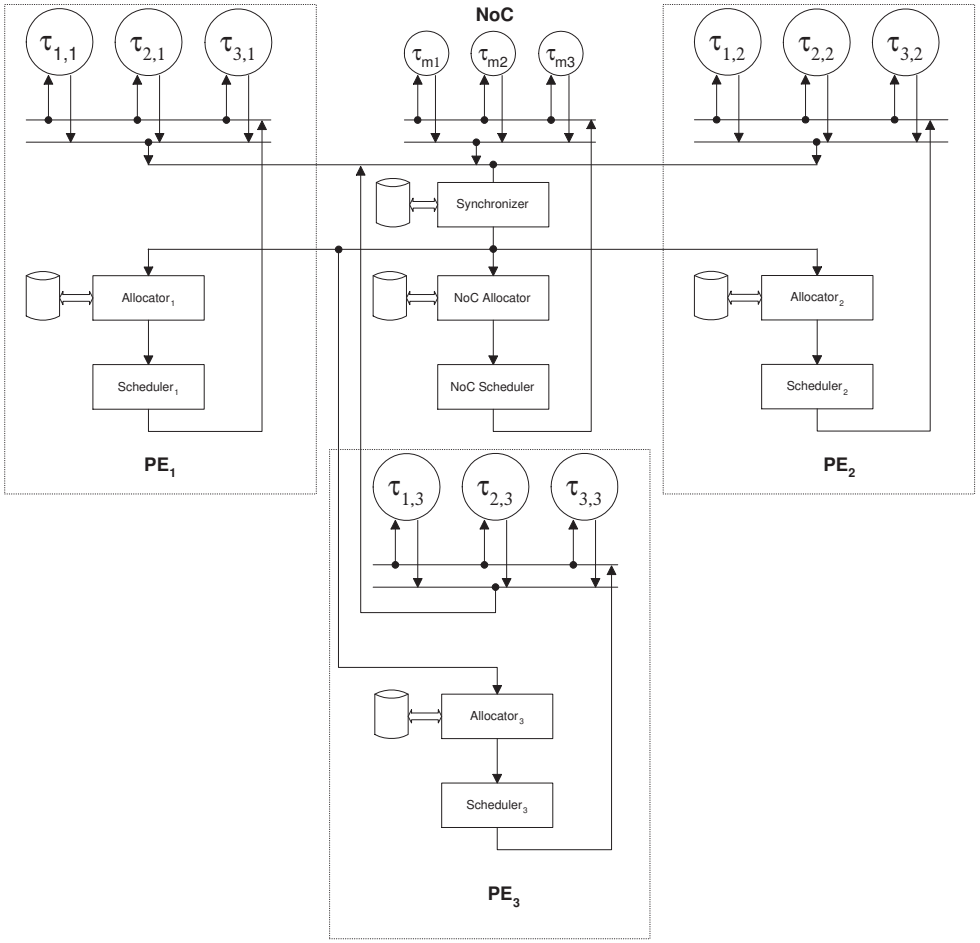


Fig. 2. *The Network-on-Chip model.*

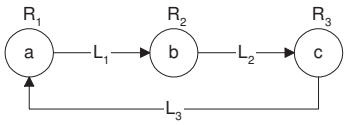
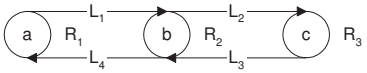
Message Task	Path					
		Torus			Mesh	
		Resource Allocation	Scheduling Needs		Resource Allocation	Scheduling Needs
			Small Message Size	Large Message Size		Small or Large Message Size
τ_{mx}	a→b	L_1	Immediate	Preemptive	L_1	Immediate
τ_{my}	c→b	L_3, R_1, L_1	Immediate	Immediate	L_3	Immediate

TABLE I
A sample reservation for two sample networks.

on all its resources. In a segmented network, these resources are laid-out as two-dimensional interconnects and are a collection of nodes (routers) and links. The NoC allocation and scheduling algorithms map a τ_m onto the available network resources. Here, we mainly illustrate this for the networks which allow parallel communication to occur, such as the segmented networks.

A. NoC Allocator

The allocator translates the path requirements of a τ_m in terms of its resource requirements such as bandwidth, buffers, etc. It attempts to minimize resource conflicts. The links and nodes in a communication path are set aside dynamically (i.e., only for the requested time slot) in the resource database. If the resource reservation process is successful, the message task is queued for scheduling. The resource allocation for two sample networks is shown in Table I. If there is a contention over a resource, then resource arbitration occurs. The arbitration mechanism is based on the underlying network implementation and is discussed shortly. In this discussion, the resources are regarded as non-preemptable. Therefore, a

resource is free to be assigned to another τ_m only after the τ_m, which is already occupying that resource, has released it.

B. NoC Scheduler

The NoC scheduler executes the τ_m's according to the particular network service requirements. It attempts to minimize resource occupancy. In a network, resource occupation is dictated by the size of the message. This concept is better illustrated using the example in Table I, where the scheduling needs for two sample networks are shown. For a mesh there is no resource conflict. The τ_m's get the required resources allocated 'immediately'. But in the case of a torus, it might experience a resource conflict for the link L_1. Here, in the event of a small message size, where τ_{mx} is finished before τ_{my} asks for L_1, there is no scheduling problem. The resources can be 'immediately' assigned to the τ_m's. But in the case of a large message size, where τ_{mx} is still running when τ_{my} asks for the link L_1, resource contention occurs. Thus, the scheduling of the messages has to be performed preemptively.

Let us consider the above example from the points of view of the network-designer and the system-designer. At the network-level, seeing the resource conflict as a network problem, the network designer may over-design link L_1 by providing excess bandwidth or introduce processing overhead, such as TDM-based message interleaving. These techniques would restore fair servicing for both the τ_m's, reducing the degree of contention. However, at the system-level, it may be possible to reschedule the communication event between the PE's (either τ_{mx} or τ_{my}). This opens up the possibility of an alternate path assignment for the τ_m's or simply stalling one of the traffics until the other has passed. System designers may even realize that large message sizes (to the extent where L_1 is contentious) never occur within the system. This could save potential scheduling/computation overhead in terms of hardware real-estate, power, etc. at router R_1 and on link L_1 as was envisioned by the network designer. Thus, when seen from the system-level, a trade-off between the NoC resource allocation and scheduling would not only complement better self-utilization, but might provide other useful insights for design improvements. Towards this, we implement a NoC model for system-level evaluation.

C. Implementation

The NoC model has exactly the same structure as the abstract RTOS model but with some modifications to its constituent module blocks. The main idea while implementing the NoC model was to preserve the existing structure of the abstract RTOS framework and to reuse the existing code fragments as much as possible so that no extra complexity is added and the code size does not grow too much so as to compromise the simulation speed. The message routing scheme currently implemented in our NoC model is that of fixed routing but the framework does have provisions for implementing other routing schemes.

1) Message Task: The message task has the same FSM structure as the Task model in the abstract RTOS model with some modifications to take out preemption and introduce resource requirements. The τ_m implementation accepts a number of arguments for its characterization. The *Message Task ID* enables the Synchronizer and the NoC Scheduler to identify the τ_m sending the message. Similarly, the *NoC Scheduler ID* is meant for the τ_m's to recognize their scheduler for exchanging various control messages. The lower- and the upper-bounds on the transmission latency of an τ_m through the NoC are defined by the *BCET (Best-Case Execution Time)* and the *WCET (Worst-Case Execution Time)*. If a message task has a certain setup time before it is released, then its *offset* is non-zero. A list of resources (links, routers, etc.) required by a τ_m during its execution is furnished in the form of *Resource ID's* and the time durations for holding those resources are specified as *CSL's (Critical Section Lengths)*. The implementation of a τ_m can be viewed as a FSM that manages various counters after sending messages to the NoC Scheduler and the NoC Allocator and upon receiving commands from the NoC Scheduler.

2) NoC Allocator: The NoC Allocator manages its resource database upon receiving `request` and `release` messages from the τ_m's. The resources are allocated to the τ_m's dynamically and they are released by the τ_m's immediately after usage. This makes resource management very flexible. In this implementation, the resources are served by the NoC Allocator on a first-come-first basis but other allocation policies can be implemented as well. Whenever a requested resource is available, the NoC Allocator sends a `grant` message to the NoC Scheduler and whenever a requested resource is occupied, there is a resource contention and the NoC Allocator sends a `refuse` message to the NoC Scheduler for an appropriate action.

3) NoC Scheduler: The NoC Scheduler receives the `ready` and `finished` messages from the τ_m's through the Synchronizer and the `grant` and `refuse` messages from the NoC Allocator. It then issues the `run` and `buffer` commands to the τ_m's. Whenever a Task running on a PE, is finished and needs to communicate with a Task running on another PE, it sends a `finished` message to the Synchronizer which maintains a task dependency database and passes the `ready` message for the corresponding τ_m to the NoC Scheduler which issues the `run` command to that τ_m.

Whenever there is a resource contention, the NoC allocator issues a `refuse` message to the NoC Scheduler which then either terminates the execution of the requesting τ_m (equivalent to message dropping) or blocks the τ_m from execution (equivalent to message buffering) till the requested resource becomes available again which is indicated by the `grant` message sent by the NoC Allocator to the NoC Scheduler. The message dropping or buffering decision is taken by the NoC Scheduler according to its underlying network implementation.

V. Results

The results of our SystemC implementation of the NoC model from Figure 2 are presented in Figure 4 and Figure 5 and illustrated in Figure 6. The sample SoC-NoC setup is shown in Figure 3. The application is assumed to have been decomposed into four tasks (τ_1, τ_2, τ_3, and τ_4). Three PE's (PE_a, PE_a, and PE_a) are selected to execute these tasks.

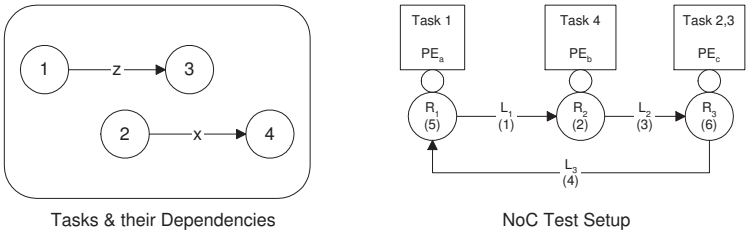
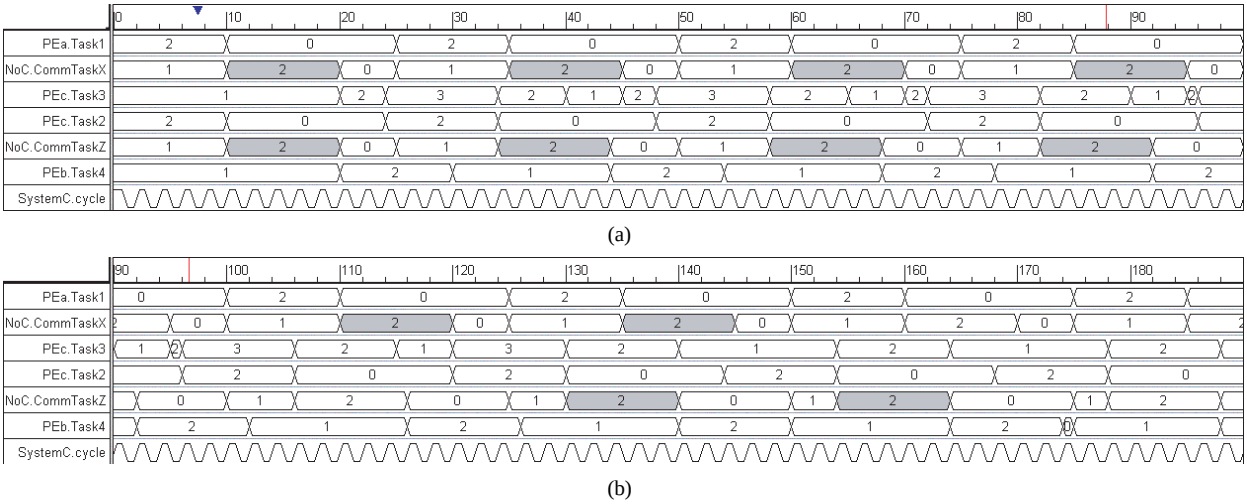


Fig. 3. *System simulation model.*



(a)

(b)

Fig. 4. *Simulation results for communication events. State enumeration: 0=inactive, 1=ready, 2=running, 3=preempted.*

The task mappings are: $\{\tau_1\} \mapsto PE_a$, $\{\tau_4\} \mapsto PE_b$, and $\{\tau_2, \tau_3\} \mapsto PE_c$. τ_2 has a higher priority than τ_3, so it can preempt τ_3 on PE_c. In this example, we look at a simple case where all the tasks are modeled identically with a period of 25 time units (except for τ_2, which has a period of 24 time units due to the priority-assignment scheme in the Rate Monotonic scheduling), an execution time (both BCET and WCET) of 10 time units and a deadline of 22 time units.

The communications between the tasks are modeled as τ_m's (as described in Section 4) which execute on a communication processor simulating a torus network using the store-and-forward routing protocol [7] (with infinite buffer at the source and the destination nodes). The message task paths and dependencies are: τ_{mx}, from PE_a to PE_c using L_1, R_2, and L_2, and τ_{mz}, from PE_c to PE_b using L_3, R_1 and L_1. Thus, the link L_1 experiences a possible contention. In our SoC-NoC test setup, the resource ID is given in brackets (next to the resource label in Figure 3). We present two cases of interest:

In Figure 4(a), modeling of two concurrent communications is shown. As mentioned earlier, there is a link contention between τ_{mx} and τ_{mz} for L_1. It is resolved by scheduling L_1 at different times among the τ_m's within the time-slot of 10 to 20 time units (and subsequent time slots). L_1 is used from 11 to 14 time units in τ_{mx} and from 17 to 20 time units in τ_{mz}. Figure 5 shows the log file of resource occupancy (Resource# 1 is link L_1). Figure 6 provides a graphical representation (Note that 1 time unit is consumed in the network setup during simulation). Thus, our model clearly supports concurrent communication as observed in segmented networks.

Figure 4(b) shows the interplay of process modeling and interconnect activity. Consider the signal titled PE_c *Task 3* (τ_3) in Figure 4(b) at a point close to the time period of 95 time units. Here, it is clear that τ_3 starts accepting the communication message and is then preempted by τ_2 on PE_c because of its higher priority. Once τ_2 is finished, τ_3 resumes and completes in time (at time 120) before its deadline. Now consider the next execution of τ_3. Both τ_2 and τ_3 are in contention. τ_3 does not even start; instead, τ_2 starts on the PE_c. τ_3, here, is not able to accept the message communicated to it by τ_1. This brings us to an interesting role of the NoC. In this simulation, we have enabled the routers to be able to buffer messages. Thus the τ_{mx} finishes freeing up its resources although τ_2 has yet to begin. τ_3, when finished, is thus able to initiate τ_{mz}, which is when τ_2 resumes.

Consider the case where the same torus network processor is running wormhole routing (plots not provided). Then, in the preemption case, the τ_{mx} stalls, holding the link L_1. As τ_2 has already preempted τ_3 on PE_c, when it is complete, it would attempt τ_{mz}. But this would not be possible as the link L_1 required here is busy in τ_{mx}, thus stalling τ_{mz}. This causes deadlock in the system. As seen earlier, we can resolve it either by introducing buffering in the routers or we have

```
 0 Initializations
10 CommTask X Released by the Synchronizer
10 CommTask Z Released by the Synchronizer
11 task x (request resource# 1)-> allocator
11 NoC_allocator (granted)->NoC_scheduler
11 task z (request resource# 4)-> allocator
11 NoC_allocator (granted)-> NoC_scheduler
14 task x (release resource# 1)-> allocator
14 task x (request resource# 2)-> allocator
14 NoC_allocator (granted)-> NoC_scheduler
14 task z (release resource# 4)-> allocator
14 task z (request resource# 5)-> allocator
14 NoC_allocator (granted)-> NoC_scheduler
17 task x (release resource# 2)-> allocator
17 task x (request resource# 3)-> allocator
17 NoC_allocator (granted)-> NoC_scheduler
17 task z (release resource# 5)-> allocator
17 synchronizer (release)-> allocator
17 task z (request resource# 1)-> allocator
17 NoC_allocator (granted)-> NoC_scheduler
20 task x (release resource# 3)-> allocator
20 task x (finished)-> scheduler 2
20 synchronizer (finished)-> allocator
20 NoC_allocator (finished)-> NoC_scheduler
20 task z (release resource# 1)-> allocator
20 task z (finished)-> scheduler 2
20 synchronizer (finished)-> allocator
20 NoC_allocator (finished)-> NoC_scheduler
   and so on...
```

Fig. 5. *Simulation log.*

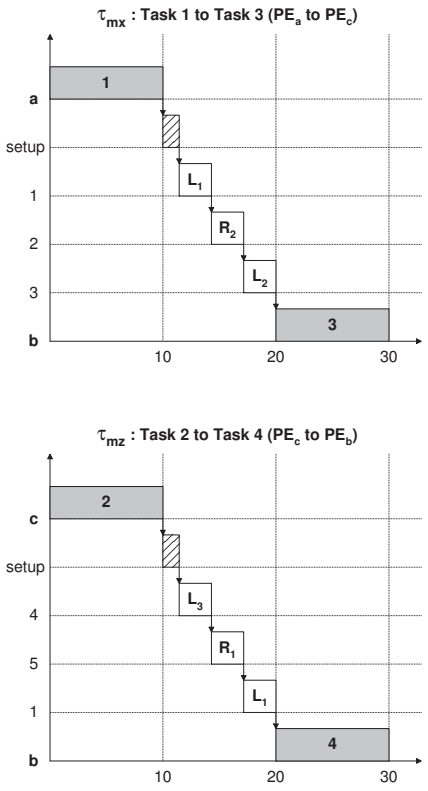


Fig. 6. *NoC allocation and scheduling for the first communication cycle.*

the freedom to choose an alternate network implementation or scheduling strategy. Thus, even this simple example clearly demonstrates the global performance evaluation for codesign when both SoC and NoC are jointly modeled.

VI. Design-Space Exploration

Figure 7 illustrates how our proposed NoC model can be used for design-space exploration at the system level. We have used three sample network topologies: torus, mesh, and bus. The assignment of tasks to the PE's are: $\{\tau_1, \tau_2\} \mapsto PE_a$, $\{\tau_3\} \mapsto PE_b$, and $\{\tau_4, \tau_5\} \mapsto PE_c$. All the tasks have the same period, execution time (BCET=WCET) and a deadline of 100, 15 and 100 time units, respectively. It is assumed that the tasks are mapped on the PE's in such a way that none of them misses its deadline. The task dependencies are: $\tau_3 \prec \{\tau_1, \tau_4\}$ and $\tau_5 \prec \tau_2$. The dependencies for the tasks mapped onto different PE's translate into τ_m's as described in Section 4. In this illustration, we have labeled them as x, y, and z. The link and the node utilization for each corresponding topology-protocol combination alters for these τ_m's. For simplicity, we model all link occupancies to be 10 time units and node processing times to be 2.5 time units. Besides, the task and the communication model, in this analysis we have also included the time spent at the network interface for message transfer from the PE's to the NoC. This is assumed to be about 3 time units. It is incurred twice, once at the source and then at the destination, for each communication event.

The three rows in Figure 7 show the network performance for three different scheduling-architecture combinations. The performance of the system is judged by its scheduling. In the first row, basic timing-aware scheduling is illustrated. Here, the networks are quite primitive, i.e., the link contention is resolved randomly. The best-effort scheduling for the torus network and the bus consumes about 80 time units. The mesh network utilizes 65 time units. The bus is a singular entity and, hence, the NoC allocator does not have much freedom in its allocation. The scheduling of the communication is, therefore, sequential. On the other hand, the torus and the mesh networks have multiple ways to allocate and schedule their resources. As the example is relatively small, therefore, the full potential of concurrent communication is not obvious for the torus network. But it is obvious for the mesh network. It is about 10 time units less than the other networks. Regarding the link utilization[1], both for the torus and the mesh networks, one link each is not used in this architectural setup (L_1 for the torus and L_4 for the mesh network). Thus, if the system is not under the constraints of meeting the timing-bounds, a possible network optimization exists. On the other hand, in the torus network, if τ_1 and τ_4 are scheduled together, there is a contention on link L_1, so a network optimization to meet the timing-bounds is required.

In the second row of Figure 7, we illustrate one possible network optimization, namely, the effects of source-based QoS routing. Any traffic from PE_a is considered to have a higher priority and, hence, is assigned the contentious resource (when

[1]Link Utilization is defined as the aggregation of the number of links occupied in the smallest time unit

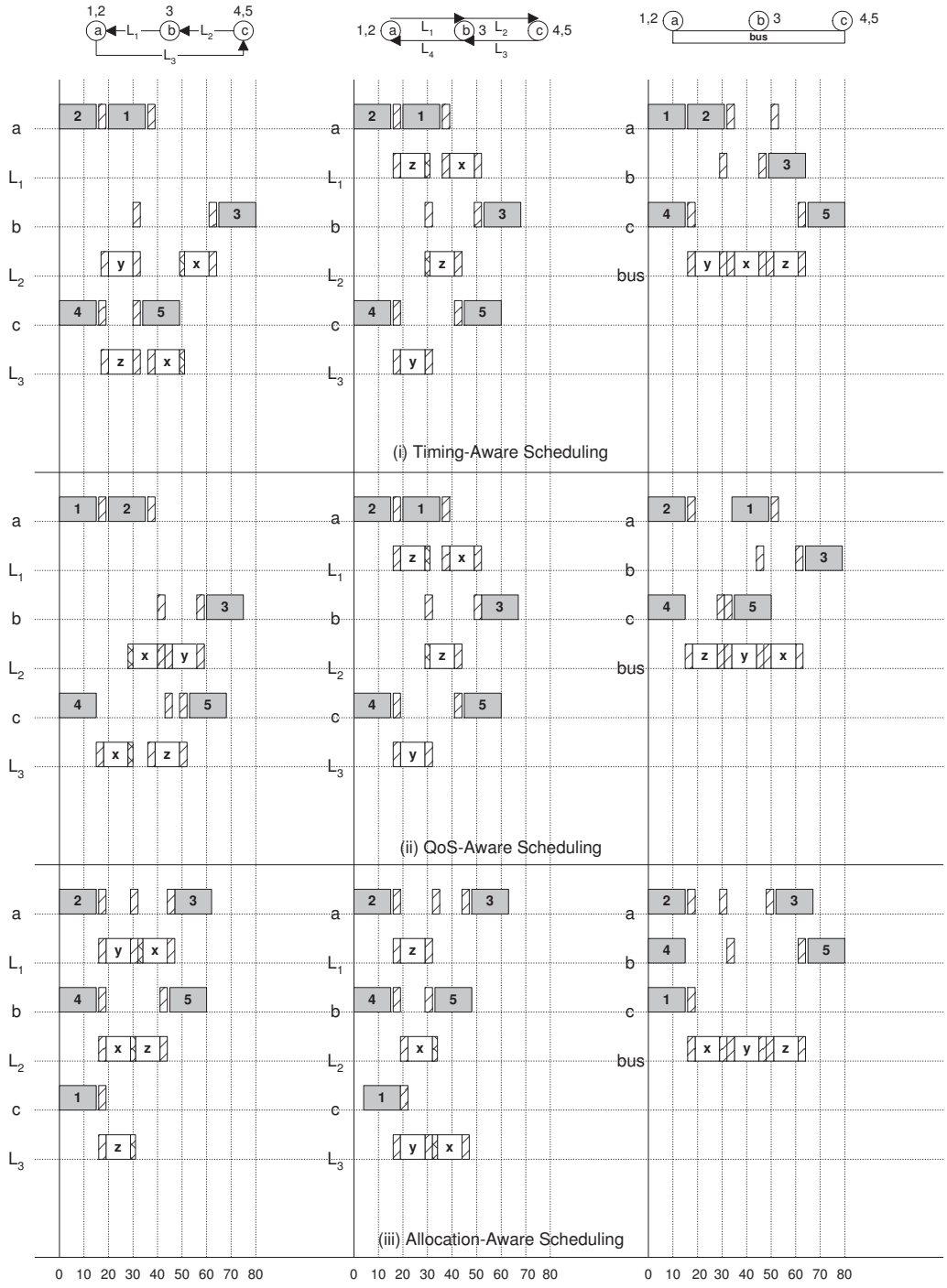


Fig. 7. *Illustration of the system-level design-space exploration.*

necessary). For a mesh network, there is no effect as the link occupancy is not in conflict. But consider its effect on the torus network. It gives about 5 time units better performance than the regular torus network. For a complex system with multiple links and nodes and handling numerous messages, these advantages are expected to be significant (both for torus and mesh). The bus architecture, on the other hand, would become a bottleneck in communication.

Having looked at how a manipulation of the network affects the overall performance, at the system-level, one can even expect to change the allocation of tasks based on the network choice. This is illustrated in the last row. The new allocation under consideration is: $\{\tau_2, \tau_3\} \mapsto PE_a$, $\{\tau_4, \tau_5\} \mapsto PE_b$, and $\{\tau_1\} \mapsto PE_c$. The advantage in terms of overall system execution time is considerable for the segmented network compared to the bus. The reasons for the poor performance of

the bus are the same as the ones stated earlier. In the case of the torus and the mesh networks, the link utilization is high now. Many links, though not all, are used simultaneously without any contention. We have not considered QoS assignment in this case, but its effect on performance, especially, in a large system might be considerable.

Using these illustrations, similar analysis for memory and power utilization can be easily performed as well. There are many possibilities of trade-offs during each iteration; namely to change the resource requirements, resource allocation, or scheduling. The overall idea is to assist the codesign process to converge while satisfying the desired performance criteria.

VII. Conclusions

We have presented an abstract modeling framework based on SystemC which supports the modeling of multiprocessor-based RTOS's and their interconnection through a NoC. The aim is to provide the system designer of single-chip, real-time embedded systems with a simple modeling and simulation framework in which one can experiment with different task mappings, RTOS policies and NoC structures and protocols in order to study the consequences of local decisions on the global system behavior and performance. We have presented how our initial multiprocessor RTOS model has been extended to handle NoCs. So far, our experimental work has been aimed at providing a proof-of-concept as demonstrated in Section 5. We are currently working on extending the NoC model to incorporate issues like, dynamic path routing, packet switching and power profiling. We are also working on a few large real-life examples as well as a schedule viewer based on the output from the monitors which will provide detailed and annotated views of the system behavior such as detailed network usage and power- and memory-profiles.

References

[1] A. Baghdadi and N-E. Zergainoh, "Design Space Exploration for Hardware/Software Codesign of Multiprocessor Systems," in *Proceedings of the 11th International Workshop on Rapid System Prototyping (RSP)*, 2000, pp. 8 – 13.

[2] L. Benini and G. D. Micheli, "Network on Chips: A New SoC Paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70 – 78, January 2002.

[3] S. H. Bokhari, "Communication Overhead on the Intel Paragon," NASA Langley Research Center," NASA Contractor Report 1982(11), September 1995.

[4] W. Brainbridge and S. Furber, "Delay Insensitive System-on-Chip Interconnect using 1-of-4 Data Encoding," in *International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2001, pp. 118 – 126.

[5] A. S. Cassidy, J. M. Paul, and D. E. Thomas, "Layered, Multi-Threaded, High-Level Performance Design," in *Design Automation and Test in Europe, DATE*, March 2003, pp. 954–959.

[6] J. Cong, "An Interconnect-Centric Design Flow for Nanometer Technologies," in *International Symposium on VLSI Technology, Systems, and Applications*, 1999, pp. 54 – 57.

[7] D. E. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan-Kaufmann, 1998, 1st edition.

[8] A. Gerstlauer, H. Yu, and D. Gajski, "RTOS Modelling for System-Level Design," in *Design Automation and Test in Europe, DATE*, March 2003, pp. 132–137.

[9] M. J. Gonzalez and J. Madsen, "Abstract RTOS Modeling in SystemC," in *Proceedings of the 20th IEEE NORCHIP Conference*, November 2002, pp. 43 – 49.

[10] P. Guerrier and A. Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections," in *Design Automation and Test in Europe, DATE*, March 2000, pp. 250 – 256.

[11] R. Ho and K. W. Mai, "The Future of Wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490 – 504, April 2001.

[12] J-M. Daveau, T. B. Ismail, and A. A. Jerraya, "Synthesis of System-Level Communication by an Allocation-Based Approach," in *Proceedings of the 8th International Symposium on System Synthesis (ISSS)*, September 1995, pp. 150 – 155.

[13] A. Jantsch and H. Tenhunen, *Networks on Chip*. Kluwer Academic Publishers, 2003.

[14] P. V. Knudsen and J. Madsen, "Integrating Communication Protocol Selection with Hardware/Software Codesign," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 8, pp. 1077 – 1095, 1999.

[15] S. Kumar, A. Jantsch, J-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A Network-on-Chip Architecture and Design Methodology," in *IEEE Computer Society Annual Symposium on VLSI*, April 2002, pp. 117 – 124.

[16] J. Madsen, K. Virk, and M. Gonzalez, "Abstract RTOS Modelling for Multiprocessor System-on-Chip," in *International Symposium on System-on-Chip*, November 2003.

[17] P. Mattson, W. J. Dally, S. Rixner, U. J. Kapasi, and J. D. Owens, "Communication Scheduling," in *International Conference on Architectural for Programming Languages and Operating Systems*, 2000.

[18] G. D. Micheli, R. Ernst, and W. Wolf, *Readings in Hardware/Software Co-Design*. Morgan-Kaufmann, 2001, 1st edition.

[19] V. J. Mooney and D. M. Blough, "A Hardware-Software Real-Time Operating System Framework for SoC's," *IEEE Design & Test of Computers*, vol. 19, no. 6, pp. 44 – 51, Nov/Dec 2002.

[20] J. Sifakis, "Modelling Real-Time Systems - Challenges and Work Directions," in *EMSOFT, Lecture Notes in Computer Science 2211*, October 2001.

[21] J. Sun and J. Liu, "Synchronization Protocols in Distributed Real-Time Systems," in *Proceedings of the 16th International Conference on Distributed Computing Systems*, May 1996, pp. 38 – 45.

[22] SystemC Workgroup, "http://www.systemc.org."

[23] T. T. Ye, L. Benini, and G. D. Micheli, "Packetized On-Chip Interconnect Communication Analysis for MPSoC," in *Design Automation and Test in Europe, DATE*, March 2003, pp. 344 – 349.

[24] X. Zhu and S. Malik, "A Hierarchical Modeling Framework for On-Chip Communication Architectures," in *International Conference on Computer-Aided Design (ICCAD)*, 2002, pp. 663 – 670.

Chapter 5

A System-Level Multiprocessor System-on-Chip Modeling Framework

Kashif Virk and Jan Madsen. A System-Level Multiprocessor System-on-Chip Modeling Framework. *Proceedings of the IEEE International Symposium on System-on-Chip* (SoC'04), November 2004. Pages: 81-84. Published.

A System-level Multiprocessor System-on-Chip Modeling Framework

Kashif Virk and Jan Madsen
System-on-Chip Group
Computer Science & Engineering Section
Department of Informatics & Mathematical Modeling
Technical University of Denmark, Lyngby 2800, Denmark
email: {virk, jan}@imm.dtu.dk

***Abstract*—We present a system-level modeling framework to model system-on-chips (SoC) consisting of heterogeneous muliprocessors and network-on-chip communication structures in order to enable the developers of todays SoC designs to take advantage of the flexibility and scalability of network-on-chip and rapidly explore high-level design alternatives to meet their system requirements. We present a modeling approach for developing high-level performance models for these SoC designs and outline how this system-level performance analysis capability can be integrated into an overall environment for efficient SoC design. We show how a hand-held multimedia terminal, consisting of JPEG, MP3 and GSM applications, can be modeled as a multiprocessor SoC in our framework.**

I. Introduction

Networks on chip (NoC's) are receiving considerable attention as a solution to the interconnect problem in highly-complex chips. The reason is two-fold. First, NoC's help resolve the electrical problems in new deep-submicron technologies, as they structure and manage global wires. At the same time, they share wires, lowering their number and increasing their utilization. NoC's can also be energy-efficient and reliable, and are scalable compared to buses. Second, NoC's also decouple computation from communication, which is essential in managing the design of billion-transistor chips. NoC's achieve this decoupling because they are traditionally designed using protocol stacks, which provide well-defined interfaces separating communication service usage from service implementation. Using networks for on-chip communication when designing systems-on-chip (SoC), however, raises a number of new issues that must be taken into account. This is because, in contrast to existing on-chip interconnects (e.g., buses, switches, or point-to-point wires), where the communicating modules are directly connected, in a NoC, the modules communicate remotely via network nodes. As a result, interconnect arbitration changes from centralized to distributed, and issues like out-of order transactions, higher latencies, and end-to-end flow control must be handled either by the intellectual property block (IP) or by the network.

Multimedia is an increasingly important application area for NoC platforms, in particular, for the new generations of hand-held devices where high-quality audio and video have to be delivered under strict resource and energy constraints. Baiceanu et al. [1] have analyzed the consequences of applying rate-monotonic (RM) scheduling on multimedia applications, i.e. an MPEG player. They argue that the complexity and dynamic behavior of this type of application makes static solutions infeasible and, hence, adaptive methods have to be used. [2] presents a more extensive survey of OS support, and, in particular, scheduling methods for multimedia applications. The presented methods are discussed in the context of basic system requirements for multimedia. In [3], Nieh and Lam present an integrated processor scheduling algorithm for multimedia applications, where both audio and video streams have to be manipulated within well-defined timing requirements, whereas conventional interactive and batch activities still have to be handled. The scheduling algorithm uses two different scheduling policies within the same scheduler, i.e., multimedia tasks are handled by an EDF scheduling algorithm, whereas conventional tasks are scheduled by a Round-Robin scheduling algorithm. The approach of having several scheduling policies within the same scheduler is further explored by Goyal et al. in [4]. They present a framework for hierarchical CPU scheduling in which different scheduling algorithms are employed for different parts of a multimedia application in order to better support the variety of best-effort, hard, and soft real-time characteristics which are typically found in multimedia computing environments. In [5], the scheduling of audio and video multimedia applications is brought to multiprocessor systems. Although a multiprocessor scheduling algorithm has been presented, the network communication latencies have not been taken into account.

In this paper, we present a system-level NoC model, which is an extension of our previous multiprocessor SoC modeling framework [6]. The extended model is able to model heterogeneous multiprocessor architectures interconnected through a an on-chip network architecture, such as a mesh or a torus. We show how a hand-held multimedia terminal, consisting of integrated JPEG encoding and decoding, and MP3 decoding as well as GSM encoding and decoding for the wireless transmission, can be modeled at the system-level in our modeling framework.

II. System-Level Modeling

To address the system-level design challenges described above, we need an extended system-on-chip design process, including the effects of the network-on-chip, with the ability to evaluate options and make critical architectural decisions based on a system-level representation in advance of a detailed design. A key pre-requisite is a library of abstract component

models that captures their respective performance, power, and physical characteristics.

The primary goal of system-level modeling for embedded systems is to formulate a model within which a broad class of designs can be developed and explored. Moreover, the difficulty of verifying the design of complex systems can be reduced by decomposing a system into smaller subsystems, independently verifying an implementation of the subsystems, and then proving that the composition of the subsystem specifications satisfies the overall system specification. In order to do so, accurate modelling of the system and all the interrelationships among the diverse processors, software processes, physical interfaces and interconnections is needed.

The scheduling problem, central to the analysis of the complexity of concurrent programs, depends on the way in which the scheduled tasks are mapped on the processing elements which, in turn, is linked with the physical architecture of the computing platforms.

A real-time operating system is meant to provide some assurances about the timely performance of tasks. Unfortunately, most mechanisms used in the basic RTOS services are not compositional in nature. Even if a mechanism can provide assurances individually to each task, there is no systematic way to provide assurances for an aggregate of two except in trivial cases.

To support the designers of single chip-based embedded systems, which includes multiprocessor platforms running dedicated RTOS's, we have developed a modeling environment based on SystemC [6], [7]. In our abstract RTOS modeling framework, we deal with generalized abstract tasks, processing elements, and communication infrastructures. For the purposes of modelling, three distinct but closely-related RTOS services have been identified, namely, task scheduling, execution synchronization, and resource allocation.

III. Model Implementation

We have implemented our system-level modeling framework in SystemC. SystemC is in a class of languages that target modeling of hardware and software systems, and it has the desirable feature of being able to simulate models at a very high level of abstraction together with low-level ones. Figure 1 gives an overview of our system-level SoC model, including the processor model and the NoC model which will be described in this section.

A. Abstract RTOS Model

Our abstract RTOS System Model [7] deals with the analysis of the execution behavior of a real-time application running on a heterogeneous multiprocessor platform. In our model, such an application is represented as a multi-threaded application comprising a set of tasks where each task, τ, can be decomposed into a sequence of task segments, τ_i. Each task segment, τ_i, is required to precede a given set of other task segments. Moreover, each task segment also excludes a given set of other task segments for the use of shared resources. For each task, we are given a release time, r_k, a release-time offset, o_i, a start time, s_k, a best-case execution time, $bcet_i$, a worst-case execution time, $wcet_i$, a deadline, d_i, a period, T_i, and a context switch time, csw_i. A similar set of parameters can be computed for each task segment, τ_i, relative to the beginning of the task containing that task segment. The multiprocessor platform is modelled as a collection of Processing Elements, PE_k, and Devices, D_k, interconnected by a set of Communication Channels, C_k. Each PE_k is modelled in terms of the RTOS services provided to the tasks comprising the application. Based on the principle of composition, three basic RTOS services are modeled: a scheduler, a synchronizer, and a resource allocator.

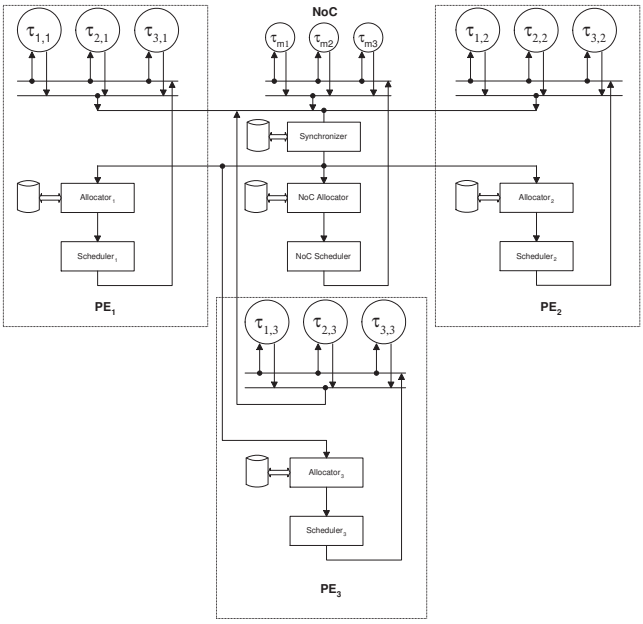


Fig. 1. **System-level System-on-Chip model**

The scheduler is modeled around the priority-based preemptive scheduling policy which is one of the most preferred scheduling policies for the execution of tasks in real-time systems due to its higher schedulability. According to our scheduler model, whenever a task becomes ready or finishes execution, the scheduler is called and it then looks for a ready task with maximal priority to continue execution. In our synchronizer model, synchronization is regarded as a means to prevent undesirable task interleavings by the scheduler. Our synchronizer model is responsible for establishing the correctness of the results computed by the multiprocessor platform and it implements the Direct Synchronization (DS) protocol [8].

B. Extension of the Abstract RTOS Model to Model NoCs

For the purpose of forming a system-level NoC simulation model, unlike a network simulator, we have abstracted away all the low-level network details except the most essential ones (e.g., topology, latency, etc.). We treat the on-chip communication network as a *communication processor* to reflect the servicing demands. A communication event within this network is modeled as a *message task*, τ_m, executing on the communication processor. When one PE wants to communicate with another PE, a τ_m is fired on the communication

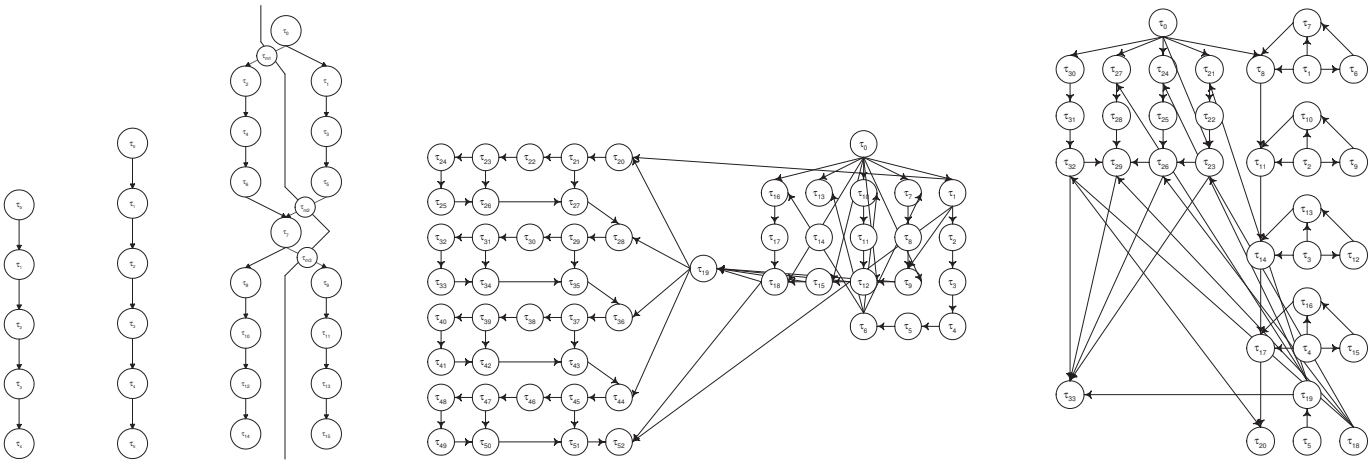


Fig. 2. **The five task graphs corresponding to the multimedia applications. From left, these are the JPEG Encoder, JPEG Decoder, MP3 Decoder, GSM Encoder and the GSM Decoder.**

Application Type	Number of Tasks	Deadline	Processor	Clock Frequency	Scheduler
JPEG Encoder	5	250ms	GPP0	25MHz	Rate Monotonic
JPEG Decoder	6	500ms	GPP0	25MHz	Rate Monotonic
MP3 Decoder	16	25ms	GPP0	25MHz	Rate Monotonic
GSM Encoder	53	20ms	GPP0	25MHz	Earliest Deadline First
GSM Decoder	34	20ms	GPP1	10MHz	Earliest Deadline First

TABLE I
Parameters for the multimedia applications

processor. Each τ_m represents communication only between two fixed set of predetermined PE's. Since a NoC supports concurrent communication, τ_m's need to be synchronized, allocated resources and scheduled accordingly. This is a property of the underlying NoC implementation, where the NoC allocator reflects the topology and the NoC scheduler reflects the protocol. A resource database, which is unique to each NoC implementation, contains information on all its resources. In a segmented network, these resources are laid-out as two-dimensional interconnects and are a collection of nodes (routers) and links. The NoC allocation and scheduling algorithms map a τ_m onto the available network resources.

- *NoC Allocator:* The allocator translates the path requirements of a τ_m in terms of its resource requirements such as bandwidth, buffers, etc. It attempts to minimize resource conflicts. The links and nodes in a communication path are set aside dynamically (i.e., only for the requested time slot) in the resource database. If the resource reservation process is successful, the message task is queued for scheduling.
- *NoC Scheduler:* The NoC scheduler executes the τ_m's according to the particular network service requirements. It attempts to minimize resource occupancy. In a network, resource occupation is dictated by the size of the message.

IV. Hand-held Multimedia Terminal

In this section, we will demonstrate the capabilities of our system-level modeling framework by presenting the simulation results of a multiprocessor SoC-based multimedia device which concurrently runs JPEG encoding/decoding, MP3 decoding, and GSM encoding/decoding all in real-time. Figure 2 shows the five task graphs which are defining the core functionality of our multimedia device. The pre-processing steps for abstracting the application code, like the extraction of the static task graph parameters through code profiling, and mapping the task graphs to the NoC architectures have been performed manually [9]. For the purpose of demonstrating the capabilities of our modeling framework, the applications have been mapped on four processing elements (see Figure 3), three fast processors (25 MHz), and one slow processor (10 MHz). Each of the four processors has its own local memory and all the four processors are interconnected by a torus network. Using distributed memory for instructions and data greatly reduces the traffic in the network.

The MP3 decoder is the most critical multimedia application and mapping its task graph on a single processor, even on a fast processor, reveals that some tasks miss their deadlines. Therefore, the MP3 application task graph has been partitioned and mapped on two fast processors which, as mentioned above, are interconnected through a NoC. The JPEG encoder and decoder applications are mapped to the same two fast processors as the MP3 decoder, whereas the GSM encoder is mapped onto a third fast processor and the GSM decoder is mapped on a slow processor. This mapping results in the exchange of communication messages between the two fast processors over the NoC.

In order to illustrate the capabilities of our modeling framework, we are using two different schedulers. RM scheduling is used on the two fast processors to handle JPEG and MP3,

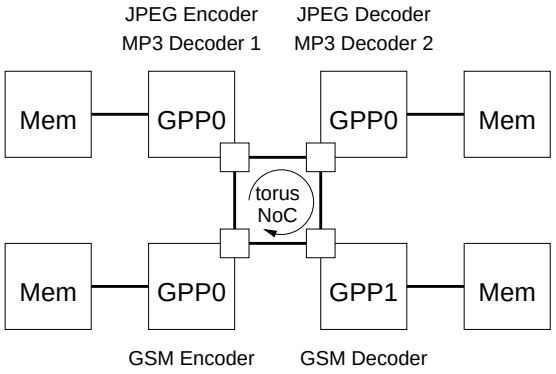


Fig. 3. **Multiprocessor architecture for the multimedia application.**

whereas the two GSM applications are scheduled using EDF scheduling. Table I summarizes the characteristics of the multimedia application.

V. Conclusions

We have presented a system-level, system-on-chip modeling framework and discussed how our original SoC model has been extended to handle the effects of the on-chip interconnection infrastructure, i.e., the network-on-chip. We have demonstrated the capabilities of our modeling framework by modeling and simulating a hand-held multimedia terminal application mapped on a heterogeneous 4-processor SoC architecture interconnected through a torus on-chip network topology. It is worth mentioning, however, that our system-level modeling framework supports more sophisticated scheduling policies and NoC topologies. Moreover, features like including the effects of the network interface and memory accesses as well as dynamic load balancing support can be built upon by adding more components to the existing framework components. We are currently extending our modeling framework to include radio and transducer components in order to be able to model wireless sensor networks, i.e., a distributed system of SoCs.

Acknowledgements

The work presented in this paper has been funded by the SoC Mobinet Project (IST 2000-30094).

References

[1] V. Baiceanu, C. Cowan, D. McNamee, C. Pu, and J. Walpole, “Multimedia Applications Require Adaptive CPU Scheduling,” in *Proceedings of the Workshop on Resource Allocation Problems in Multimedia Systems*, December 1996.

[2] J. Regehr, M. B. Jones, and J. A. Stankovic, “Operating System Support for Multimedia: The Programming Model Matters,” Microsoft Research, Tech. Rep., September 2000.

[3] J. Nieh and M. S. Lam, “Integrated Processor Scheduling for Multimedia,” in *Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, July 1995.

[4] P. Goyal, X. Guo, and H. Vin, “A Hierarchical CPU Scheduler for Multimedia Operating Systems,” in *Proceedings of the Second Symposium on Operating System Designs and Implementations (OSDI’96)*, October 1996, pp. 107–122.

[5] J. Nieh and M. S. Lam, “Multimedia on Multiprocessors: Where’s the OS When You Really Need It?” in *Proceedings of the Eighth International Workshop on Network and Operating System Support for Digital Audio and Video*, July 1998, pp. 103–106.

[6] J. Madsen, K. Virk, and M. Gonzalez, “Abstract RTOS Modelling for Multiprocessor System-on-Chip,” in *International Symposium on System-on-Chip*, November 2003, pp. 147–150.

[7] M. Gonzalez and J. Madsen, “Abstract RTOS Modeling in SystemC,” in *Proceedings of the 20th IEEE NORCHIP Conference*, November 2002, pp. 43 – 49.

[8] J. Sun and J. Liu, “Synchronization Protocols in Distributed Real-Time Systems,” in *Proceedings of the 16th International Conference on Distributed Computing Systems*, May 1996, pp. 38–45.

[9] M. Schmitz, B. Al-Hashimi, and P. Eles, “A Co-Design Methodology for Energy-Efficient, Multi-Mode Embedded Systems with the consideration of Mode Execution Probabilities,” in *Design Automation and Test in Europe, DATE*, March 2003, pp. 960–965.

Chapter 6

System-Level Modeling of Wireless Integrated Sensor Networks

Kashif Virk, Knud Hansen and Jan Madsen. System-Level Modeling of Wireless Integrated Sensor Networks. *Proceedings of the IEEE International Symposium on System-on-Chip* (SoC'05), November 2005. Pages: 179-182. Published.

System-level Modeling of Wireless Integrated Sensor Networks

Kashif Virk Knud Hansen Jan Madsen

Computer Science & Engineering Section
Department of Informatics & Mathematical Modeling
Technical University of Denmark, Lyngby 2800, Denmark
email: {virk, jan}@imm.dtu.dk

***Abstract*—Wireless integrated sensor networks have emerged as a promising infrastructure for a new generation of monitoring and tracking applications. In order to efficiently utilize the extremely limted resources of wireless sensor nodes, accurate modeling of the key aspects of wireless sensor networks is necessary so that system-level design decisions can be made about the hardware and the software (applications and real-time operating system) architecture of sensor nodes. In this paper, we present a SystemC-based abstract modeling framework that enables system-level modeling of sensor network behavior by modeling the applications, real-time operating system, sensors, processor, and radio transceiver at the sensor node level and environmental phenomena, including radio signal propagation, at the sensor network level. We demonstrate the potential of our modeling framework by simulating and analyzing a small sensor network configuration.**

I. Introduction

Wireless sensor networks have emerged as a promising infrastructure for a new generation of monitoring applications. Owing to their small form-factors, ad-hoc deployment, and extended periods of unattended operation requirements, these wireless sensor networks form an extremely resource- and energy-constrained sensing, computing, and communication environment which makes the design and optimization of these systems a challenging task. In particular, the design of the sensor nodes requires a deep understanding of their various constituent components, their underlying technologies and the interactions between those components. Figure 1 shows the elements of a wireless sensor node and its hardware and software partitioning.

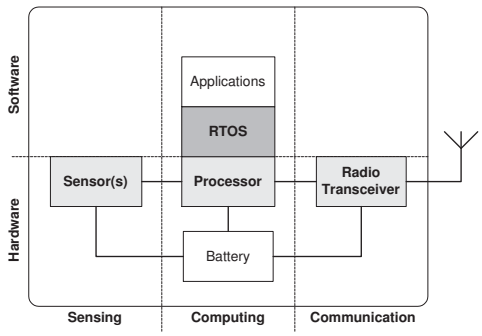


Fig. 1. *Sensor Node*

In order to be able to explore the design space at very early stages in the design process, it is important to have an accurate system-level model of the sensor network capturing all the inter-relationships among the diverse processors, software processes and radio- and sensor interfaces. In this paper, we present an extension of our earlier work on SystemC-based multiprocessor SoC modeling framework [1] which can provide the wireless sensor network designers a system-level abstraction of the sensor network for system-level design-space exploration to meet the requirements mentioned above[1].

Numerous sensor network simulators implemented in software exist, either in the open source or as commercial products, which can be broadly categorized into *improvised* sensor network simulators - based on existing network simulators or discrete-event simulation frameworks - and *custom* sensor network simulators. Typical examples of *improvised* sensor network simulators are: ns-2 [2], Opnet Wireless Module [3], and OMNeT++ [4] while common examples of *custom* sensor network simulators include: TOSSIM [5] and its extension PowerTOSSIM, Avrora [6] and its extension AEON, and Atemu [7]. Most of the *improvised* sensor network simulators emphasize sensor network level simulations (concentrating on the simulation of wireless communication protocol stacks) while a majority of the *custom* sensor network simulators focus mainly on sensor node level simulations (mostly code or processor simulations) and are either specific to certain sensor network research projects or support a limited number of sensor node platforms. A unified sensor node level as well as sensor network level simulator does not exist so far despite such attempts [8]. Moreover, to the best of our knowledge, none of the sensor network modeling approaches, reported so far, addresses the issue of designing sensor network systems from a hardware/software codesign perspective.

The main contribution of this work is to apply a HW/SW Codesign approach for the system-level modeling of a generic sensor node platform embedded in a generic sensor network environment model forming a system-level sensor network model which is fairly detailed as well as sufficiently efficient.

The rest of this paper is organized as follows: Section II provides the methodology and implementation details for our sensor network model. The results of our implementation and

[1]A part of this work was funded by the ARTIST and the Hogthrob Projects.

a simulation example elaborating our modeling framework are presented in Section III. Section IV, finally, provides conclusions and the future directions of our work.

II. Sensor Network Model

In our SystemC-based modeling framework, a sensor network model is designed following the principle of composition. We model a sensor network at two levels: the sensor network level (Figure 2) and the sensor node level (Figure 3). This section describes the details of each of these levels and their inter-relationships and interactions.

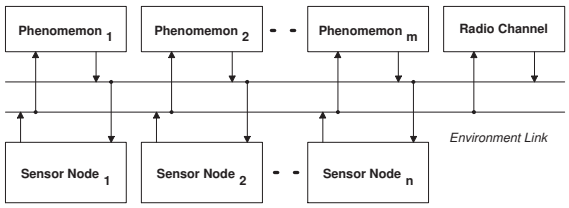


Fig. 2. *Sensor Network Level Model*

A. Sensor Node Level Model

At the sensor node level, a sensor node platform model is split into two sections: the software section - for functional simultion of the sensor node platform and the hardware section - to enable estimation of the energy consumption of the sensor node platform.

The software section of the sensor node platform model consists of the application model, comprising a set of task models, and the RTOS model, composed of a set of RTOS services [1].

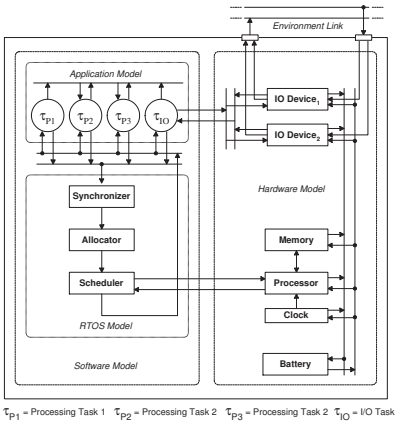


Fig. 3. *Sensor Node Level Model*

1) Application Model: The sensor node application software is modeled as a set of task models which are executed on the sensor node processor(s) under the control of RTOS(s). To accurately model the sensor node application, it is important to handle both the tasks and their possible inter-dependencies. The dependencies among the tasks are resolved by the synchronizer which is a component of the RTOS model.

The task models are the abstract building blocks from which the sensor node application model is composed. From the point of view of their activation mechanism, task models can be either *time-triggered (periodic)* or *event-triggered (sporadic)*. While periodic task models represent repetitive tasks, sporadic task models handle the response of the application model to the events that are generated either by the environment model or by other task models. In addition, from the point of view of their function or behavior, task models are organized into two groups:

- *processing task models* (τ_P) model the usage of a sensor node processor and are controlled by the RTOS model.
- *I/O task models* (τ_{IO}) model the usage of the I/O devices on a sensor node platform, e.g., the sensors and the radio transceiver. These task models form, a link between the RTOS model and the environment model with which they are interfaced using specific interface protocols (e.g., poll-based/interrupt-based, serial/parallel, etc.). There are two separate I/O tasks to model the radio transceiver behavior. The send task models radio transmission and the receive task models radio reception.

The function or behavior of a task is modeled as a finite-state machine (FSM) with five states as indicated in Figure 4: *idle*, *ready*, *running*, *preempted*, and *self-preempted*. Each task model is characterized by a set of parameters, such as the worst- and the best-case execution time, context-switching overhead, deadline, period (for a periodic task), offset, resource requirements, and precedence relations. Upon initialization, each task starts in the *idle* state and, if its offset value is zero, it transits to the *ready* state. The task remains in the *ready* state until it receives a `run` command from the RTOS scheduler upon which it transits to the *running* state. When the task has finished its execution, it issues a `finished` message to the scheduler and transits back to the *idle* state. At any time during its execution, a task may be preempted by the scheduler and it then enters into the *preempted* state where it waits till it receives a `resume` command from the scheduler which enables it to reenter the *running* state. The *self-preempted* state models the ability of an application task to release processor control to some other applicaion task requesting it, while it is waiting for an interrupt from an I/O device. Note that the *self-preempted* state is different from the *preempted* state in that the task itself controls its transition to and from it, while the transition to and from the *preempted* state is controlled exclusively by the scheduler.

The occurence of an interrupt is modeled by the `self-resume` message from a task in the self-preempted state. To service the interrupt, the priority of the self-preempted task is updated to the maximum level when it self-resumes. Thus, an interrupt is handled by the RTOS scheduler by interrupting the execution of whatever task is running at the time of its occurrence to service the interrupt and the portion of the application task running after self-resumption represents interrupt servicing. The only difference between running a high-priority task and interrupt servicing is that a high-priority task may not preempt a running task if it has the same priority, while interrupt servicing does preempt a running task, even

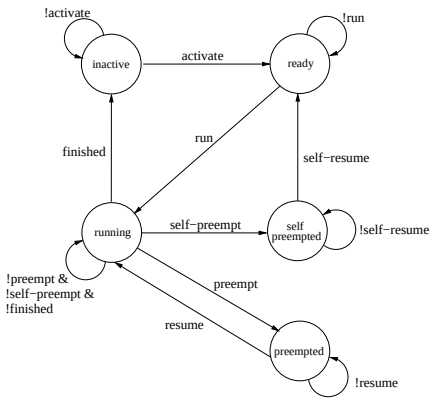


Fig. 4. *Task Model*

the interrupt servicing of another, previously-occured interrupt (e.g., in case of nested interrupts).

2) RTOS Model: The RTOS model is composed of three independent modules that model the basic RTOS services. The model is designed such that any of the RTOS services can be changed in a simple and straight-forward manner. Each module handles its relevant data independently of the other to preserve composability. A scheduler models a real-time scheduling algorithm. A synchronizer models dependencies among tasks. An allocator models the mechanism of resource sharing among tasks. For details on the model and how it is implemented in SystemC, we refer to [1].

All the task models are connected to the RTOS model through a pair of SystemC master/slave ports. In addition to that, the I/O task models are connected to the master/slave ports of the sensor node platform model which, in turn, are connected, in a similar way, to the components of the environment model. The receive and the sense task models also have activation ports (see Figure 3).

3) Battery Model: The hardware section of the sensor node platform model contains energy macro models[2] for the processor, memory, clock, and I/O devices alongwith a battery model. The battery model handles the energy consumption of a sensor node. It is connected to each of the hardware component models of the sensor node and decreases its energy resources depending on their power draw. At each clock cycle, the battery model updates it energy resources according to a certain specified function depending on the selected battery model (simplistic linear battery discharge models as well as more advanced battery models, which take the hysteresis phenomenon into account, can be selected). The link between the hardware component models is bidirectional which enables modeling the demise of a sensor node when its battery runs out of energy. The battery model can also inform the hardware component models when its energy resources go below predefined thresholds.

[2]The energy macro modeling approach refers to the pre-characterization of a hardware or a software macro-block in terms of its energy consumption using empirical, simulation, or analytical models. A macro-block comprising a system can be defined at any level of abstraction by trading-off accuracy with efficiency or vice vera, e.g., a hardware macro-block can be defined at the RT-level or a software macro-block can be defined at the instruction-level.

B. Sensor Network Level Model

At the sensor network level, a sensor node platform model is embedded in an environment model that models the environmental phenomena to be sensed by the sensor network application.

1) Environment Model: The environment model represents an abstraction of the environment as observed at the outputs of the sensors on the sensor nodes. It is composed of different component models each of which corresponds to the phenomenon monitored by the sensor network application. The environment model connects all the instantiations of the sensor node model – any of which can request it for data pertaining to a certain phenomenon. The environment model can also generate events for any instantiation of the sensor node model.

To model sensing, an I/O task model requests or gets events from the environment model component corresponding to the phenomenon (temperature, movement, etc.) according to a certain interface protocol (poll-based/event-based, serial/parallel, etc.). The receiver part of the radio transceiver is treated as a special kind of sensor and the transmitter part as a special kind of actuator. Thus, the radio signal propagation through the environment is treated as a special kind of phenomenon. The radio channel model, therefore, forms a special component of the environment model.

III. EXAMPLE

This section describes an example illustrating the capabilities of our sensor network model to capture the mechanism of radio communication among the sensor nodes. The example configuration consist of 5 sensor nodes, two of which are transmitting a message while the rest are receiving it (see Figure 5).

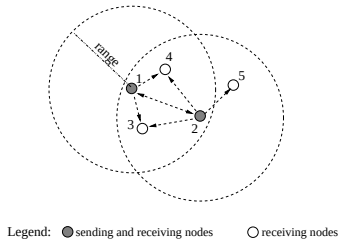


Fig. 5. *Example Topolgy*

On each sensor node, the processor runs the I/O tasks modeling the communication protocol. The transmission component of the communication protocol is described in Figure 6 and the reception component of the communication protocol is described in Figure 7. Two I/O task models have been instantiated for this example. The send task is a low-priority task, i.e., it does not preempt a running task when it initially starts. Once it has started, it periodically self-preempts and self-resumes. Everytime the send task enters its 'running' state, it steps through the states of the send protocol, either causing transition(s) to the next state(s) of the send protocol or retaining its existing state. The receive task is a high-priority task (its activation is based on the timer interrupts). Similar to

the send task, the receive task executes the receive protocol in its 'running' state.

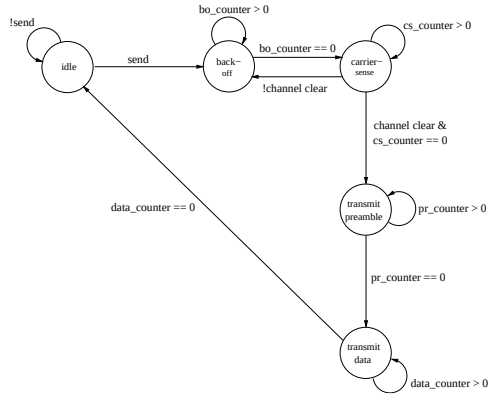


Fig. 6. *Send Protocol running on Send Task Model*

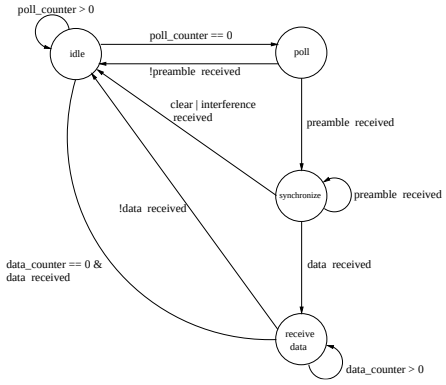


Fig. 7. *Receive Protocol running on Receive Task Model*

The simulation output waveforms corresponding to the example are presented in Figure 8. This figure represents the state of each task in terms of processor occupation as well as in terms of the communication protocol state for the send and the receive tasks. The example illustrates the behavior of the MAC (CSMA protocol) in the case of channel contention. The send task of the sensor node 2 fails to obtain channel access at its first attempt (because it detects that the sensor node 1 is transmitting). It, therefore, backs-off for a random period of time before reattempting to gain access to the radio channel. On its second attempt, the transmission of the sensor node 1 has finished and the radio channel is clear, so the sensor node 2 can send. The reason why the sensor node 1 gains access to the radio channel first is because its initial back-off time was smaller than that of the sensor node 2. Furthermore, notice that once the send task of the sensor node 1 has finished transmitting, the receive task of the sensor node 1 polls the radio channel, detects the preamble from the sensor node 2 and receives the packet sent by it.

IV. Conclusions

We have presented a system-level wireless sensor network modeling framework based on SystemC. The aim of our modeling framework is to provide designers of sensor networks

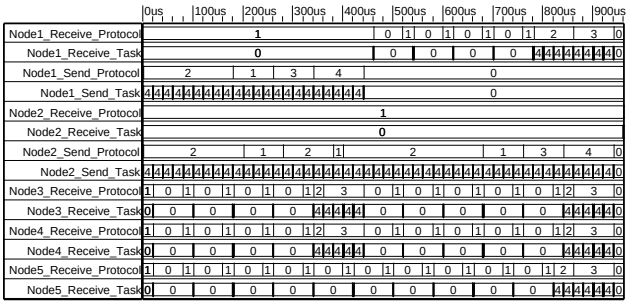


Fig. 8. *Simulation results for Example (see Table I for state enumerations).*

TABLE I

State No.	Send/Receive Task	Send Protocol	Receive Protocol
0	inactive	idle	idle
1	ready	back-off	poll
2	running	carrier-sense	synchronize
3	preempted	transmit preemble	receive data
4	self-preempted	transmit data	

with a simple modeling and simulation framework in which one can experiment with different application task mappings, RTOS policies and communication protocols in order to efficiently utilize the limited resources available. Using this framework, one can also study the consequences of design decisions taken at the sensor node-level on the behavior and performance of the sensor network. We are currently working on extending our modeling framework to incorporate more accurate power modeling. This will enable us to estimate how different power management strategies can improve the sensor network lifetime.

References

[1] J. Madsen, K. Virk, and M. Gonzalez, "Abstract RTOS Modelling for Multiprocessor System-on-Chip," in *International Symposium on System-on-Chip*, November 2003, pp. 147–150.

[2] University of California, Berkeley, "Network Simulator-2," http://www.isi.edu/nsnam/ns/.

[3] OPNET Technologies Inc., "OPNET Wireless Module," http://www.opnet.com/products/wirelessmodule.

[4] OMNet++, "OMNet++: Discrete-Event Simulation System," http://www.omnetpp.org/.

[5] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in *Proceedings of 1st International Conference on Embedded Networked Sensor Systems (SenSys 2003)*. ACM Press, 2003, pp. 126–137.

[6] B. Titzer, D. K. Lee, and J. Palsberg, "Avrora: Scalable Sensor Network Simulation with Precise Timing," in *Proceedings of 4th International Conference on Information Processing in Sensor Networks (IFIP 2005)*, April 25-27, 2005.

[7] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, and M. Karir, "ATEMU: A Fine-Grained Sensor Network Simulator," in *Proceedings of First International Conference on Sensor and Ad Hoc Communications and Networks (SECON 2004)*, October 4-7, 2004.

[8] H. Park, W. Liao, K. H. Tam, M. B. Srivastava, and L. He, "A Unified Network and Node Level Simulation Framework for Wireless Sensor Networks," Center for Embedded Networked Sensing, UCLA, Tech. Rep., Nr. 25, September 2003.

Chapter 7

Bridging Model for a Wireless Sensor Network Modeling Framework

Kashif Virk and Jan Madsen. Bridging Model for a Wireless Sensor Network Modeling Framework. Not Published.

Bridging Model for a Wireless Sensor Network Modeling Framework

Kashif Virk and Jan Madsen
System-on-Chip Group
Computer Science & Engineering Section
Department of Informatics & Mathematical Modeling
Technical University of Denmark, Lyngby 2800, Denmark
email: {virk, jan}@imm.dtu.dk

***Abstract*—This paper describes the framework for a bridging model that links the cycle-accurate sensor node model with the system-level sensor network model [7]. This model also makes use of hierarchical channels in SystemC and Transaction-Level Modeling concepts and their support in the SystemC TLM library.**

I. Introduction

The embedded computer system designers, usually, use processor-based templates to build today's system-on-chip (SoC) designs, which contain one or more cores with considerable on-chip memory and complex communication buses. Because on-chip processor cores are often either legacy or third-party components, the designers need correct functional models to accurately track the interaction of processor core(s) with the rest of the embedded system.

The embedded hardware designers use Hardware Description Language (HDL) simulators to validate their work, but these simulators model the processor micro-architecture in too much detail to efficiently simulate complex processor cores. The embedded software designers, on the other hand, routinely use cross-development toolkits containing a cross-compiler and an instruction-set simulator (ISS) to validate functionality and assess application performance.

Thus, exploring and validating a complex SoC design requires a single, integrated hardware-software cosimulation platform. The academic research groups, as well as electronic design automation vendors, have developed numerous such platforms.

Traditional cosimulation design environments use multi-language system descriptions - HDL for hardware and C (or similar languages) for software - to construct an efficient link between event-driven hardware simulators to cycle-based ISS's. Therefore, there has been a need for a system design language that describes the functionality of both hardware and software. It must allow the system to be defined, first without making assumptions about the implementation, and then to be refined into the exact implementation with hardware and software components. It is also important to be able to use standard models of computation (MOCs) at the initial design stages. Further, one may not wish to concretely specify the communication mechanisms and instead leave it to be defined by the underlying operational semantics of the MOCs being deployed.

More recently, using C/C++ for hardware design descriptions and design flows has gained popularity because using the same language for describing hardware and software can, potentially, bridge the gap between hardware and software description languages. Using the same language also makes it possible to simulate the entire system within a single simulation engine.

SystemC is the leader in system-level modeling with C++. The SystemC approach consists of a progressive refinement of specifications. SystemC allows both applications and platforms to be expressed at sufficiently high levels of abstraction while, at the same time, enabling the linkage to hardware implementation and verification. SystemC has the potential to provide a full-fledged description of an execution platform which can serve as the target of a codesign methodology. Thus, SystemC is a viable intermediate representation language.

Compared to VHDL, with SystemC, interfaces between blocks are not simply described by signals, but by communication methods and protocols. This drastically increases the design abstraction and, thus, the design efficiency. This capability is provided via an open-source C/C++ class library that extends the capabilities of C++ by providing new mechanisms to model system architecture with hardware elements, concurrency and reactive behavior (through events). It provides several class packages for specifying hardware blocks and communication channels. The design environment specifies software algorithmically as a set of functions embedded in abstract modules that communicate with one another and with hardware components via abstract communication channels.

SystemC is not a design methodology but it proposes various layers of abstraction that are useful for specification capture in the early stages of a design flow. The design cycle starts with an abstract high-level untimed or timed functional (UTF/TF) representation that is refined to a bus-cycle accurate and then an RTL (Register Transfer Level) hardware model [1].

Currently, in the majority of industrial projects, after the specification phase, what will be the software and hardware parts constituting the future SoC (System-on-Chip) is chosen following ad hoc methods, often based on the designer experiences. Then, the development of the hardware part and the software part of the SoC is performed in two disjoined design flows. This is problematic because errors appear very late in the design process and modifying hardware/software

partitioning requires a huge amount of work. The reason is the lack of tools during the partitioning phase. Several efforts are being made to ease partitioning, by making possible the specification and simulation at system level, then refining it in an iterative way towards the final implementation.

II. Transaction-Level Modeling in SystemC

In contrast to SoC modeling, the design of embedded systems, typically, incorporates the assembly of standard HW and SW components with user-designed HW (reconfigurable logic or ASIC) and SW. As system complexity continuously rises, the proper connection of user HW and SW to the system's communication architecture becomes more and more a focus of design. As a result, the development of embedded software that is closely related to the HW will have to wait for the RTL model to be completed.

To fill this gap, recently, the Transaction Level Modeling (TLM) paradigm has been widely propagated for System-on-Chip (SoC) design. A TLM approach for embedded system design with SystemC considerably relieves designers of the task of implementing platform-specific communication protocols. By orthogonalizing system functionality and system communication, very high simulation speeds become feasible enabling fast communication architecture exploration, early embedded software development, and rapid prototype generation.

Transaction Level Modeling (TLM) is a higher modeling abstraction level, above the Bus Cycle Accurate (BCA) abstraction level, for faster simulation performance. At the TLM level, architecture IPs are modeled at a functional level and the system bus is captured as an abstract 'channel', independent of a particular bus architecture or protocol implementation. A TLM model can be used as a reference prototype of the system and for early functional system validation and embedded software development.

Transaction Level Models are bit-accurate models of a system with specifics of the bus protocol replaced by a generic bus (or channel), and where communication takes place when IPs call `read()` and `write()` methods provided by the channel interface. Since detailed timing and pin-accuracy is omitted, these models are fast to simulate and are useful for early functional validation of the system.

SystemC provides a rich set of primitives for communication and synchronization - channels, ports, interfaces, events, signals and wait-state insertion. Concurrent execution is performed by multiple threads and processes (lightweight threads) and execution schedule is governed by the scheduler. SystemC also supports capture of a wide range of modeling abstractions from high-level specifications to pin- and timing-accurate system models.

SystemC separates computation and communication by having modules and processes for computation and interfaces and channels for communication.

In SystemC, **modules** are the basic building blocks for partitioning a design. The modules control and process data. A module hides its data and algorithms from other modules. A module may have one or many processes which can run concurrently. There are three types of **processes**: `sc_method`, `sc_thread` and `sc_cthread`. The modules communicate through **channels**. The channels implement communications between modules. There are two types of channels: *primitive channels* and *hierarchical channels*. The primitive channels are, in some sense, state-less while the hierarchical channels can have internal states and control flow associated with them. As the name suggests, hierarchical channels can contain other channels, modules or processes. The **interfaces** provide a mechanism to allow independence of computation modules from the mechanisms of communication channels. The interfaces specify the signature of the operations provided by channels. A *blocking interface* implies that this interface has to be called from within an `sc_thread`, as such, the implementation of the interface is allowed to contain `wait(.)` statements. In contrast, a *non-blocking interface* cannot contain a `wait(.)` statement since it is allowed to call such an interface from within an `sc_method` which is not capable of performing the context switch that is required to implement the `wait(.)` call. A module accesses a channel through a **port** whose type is one of the interfaces implemented by the channel [2].

A key feature of SystemC 2.0 is that it introduces a set of features for generalized modeling of communication and synchronisation. In SystemC 2.0, communication can be modeled at a higher level of abstraction referred to as Transaction Level Modeling (TLM). The exchange of data between two computational components of a system is called a transaction. Communication is modeled through channels and transaction requests take place using interface method calls of these channel models without any synchronization. Unnecessary details of communication are hidden in the TLM and can be worked out later on. Using TLM results in simplified design effort and also gains simulation speed as details of the low level communication infrastructure are not present.

The TLM level emphasizes what data are transferred and from which locations but not the detailed implementation based on a specific protocol. Thus, communication among components is abstracted from the details of the implementation of the communication architecture and this enables component-reuse. In addition, simulation at this level can usually be carried out at high speed.

SystemC's TLM abstracts hardware communication from signal-level clocked protocols into untimed function calls. Like subroutine abstraction, TLM can also be applied recursively at higher and higher levels, e.g., from bus read/write transactions to burst reads/writes; to DMA transfers; to complete HW accelerator functions; and so on. This abstraction also has benefits in simulation speed. While TLM standardization for module communication has been impressive, there are still some serious challenges ahead.

A. SystemC Channels

In SystemC channels are important because they enable several concepts: Appropriate channels enable safe communication between processes. Channels, in conjunction with ports, clarify the relationships of communication (producer vs. consumer) Interfaces are important in SystemC because they enable the separation of communication from processing.

The channels come in two flavors: primitive and hierarchical. The basic premise of a channel is a class that inherits from an interface. The interface makes a channel usable with ports. In addition, channels must inherit either from `sc_prim_channel` or `sc_channel`. This distinction in these latter two base classes is one of distinct capabilities and features. In other words, `sc_prim_channel` has capabilities not present in `sc_channel` and vice versa.

1) Primitive Channels: Primitive channels are intended to provide very simple and fast communications. They contain no hierarchy, no ports, and no `sc_methods` or `sc_threads`. Primitive channels have the ability to implement the evaluate-update paradigm.

2) Hierarchical Channels: By contrast, hierarchical channels may access ports, they can have processes and contain hierarchy as the name suggests. In fact, hierarchical channels are really just modules that implement one or more interfaces. Hierarchical channels are intended to model complex communications buses such as PCI, HyperTransport, or AMBA [3]–[6].

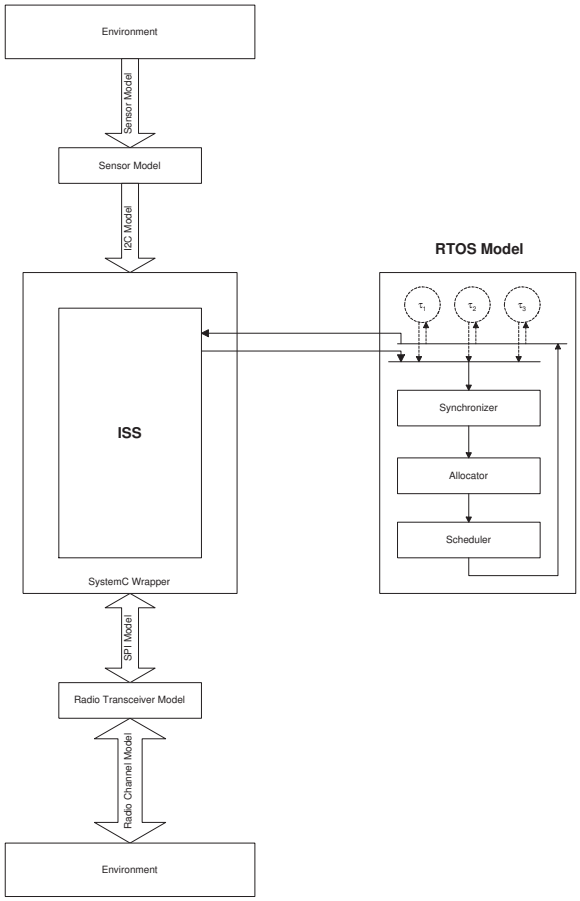


Fig. 1. Composition of Bridging Model

III. The Bridging Model

SystemC describes the functionality of both hardware and software inside a unified specification language based on C++. At a high level of abstraction, SystemC allows the use of a common language for software and hardware specifications and simulation of the whole system. However, one of the problems encountered with SystemC 2.0 is the lack of features to support embedded software modeling. For some classes of applications modeled with SystemC, it is not, currently, possible to completely model the software behavior of the targeted architecture.

The availability of RTOS models is becoming strategic inside HW/SW (hardware/software) co-design environments. An RTOS provides a very useful abstraction interface between applications with hard real-time requirements and the target system architecture. Indeed, for the simulation of software modules, such as preemption and/or priority-based scheduling, generally present in any RTOS, the SystemC simulator does not offer all the necessary functionalities. This is because, during simulation, the RTOS scheduler, responsible for determining which thread will run next, manages both software and hardware threads identically. It means that systems with hard real-time constraints requiring an RTOS (Real-Time Operating System) based on a preemptive priority-based kernel cannot be modeled in a natural manner. As a consequence, a joint refinement of the software and hardware parts is a tedious task in SystemC 2.0.

To easily simulate various hardware/software configurations, at high-level, we have succesfully developed an abstract RTOS modeling framework in SystemC by abstracting the real-time operating system features at the system level as explained in Chapters 3-4. The results have shown that simulation overhead introduced by the RTOS model is negligible while providing modeling accuracy.

A. Structure and Composition

To jointly simulate the software part with the hardware part, the bridging model refines the system-level wireless sensor network model by modeling the embedded processor by an ISS (Instruction-Set Simulator) and integrating the ISS with the RTOS model making it possible to schedule several application software modules on the ISS and to simulate, more accurately, the interaction of software with hardware.

Using an ISS abstracts away the lower-level RTL details while maintaining a reasonably-fast simulation speed. The ISS is written in C/C++, encapsulated (wrapped) in a SystemC module and simulated by SystemC that accepts a binary code obtained by the cross-compilation of the software modules. During simulation, when the scheduler relinquishes control to the ISS, the corresponding software thread with the highest priority, and ready to run, is executed. Thus, it is possible to quickly obtain a functional system model whose simulation is reliable and realistic because it depends on the actual platform architecture [1]

One of the RTOS to be modeled in detail in our modeling framework is TinyOS [7]. It offers all the advantages of a real-time operating system: a preemptive kernel, a priority based task scheduler and an interrupt system. TinyOS was selected for its low complexity, the availability of its source code and because it has successfully been ported to a range of embedded processors. The ISS model is based on the AVR processor.

The bridging model also supports a faster way of modeling communications to perform a complete simulation of a sensor network application. It supports transaction-level models of the peripheral device interfaces and the sensor and radio channel models the details of which are given below.

B. Transaction-Level Modeling of Peripheral Communication Interfaces

Instead of plugging a given peripheral directly on a system bus, it is much more easy to connect them through a serial interface whose major advantage is the reduction of communication pins. Most embedded systems comprise a set of nodes connected through field busses such as I2C, SPI, CAN, etc. In its most usual form, a node is a microcontroller connected to various sensors or actuators.

In this section we show how to model the field bus communications between the nodes of an embedded system. Whereas the methodology is generic we present it in the specific case of the SPI bus and the I2C bus.

1) SPI Interface Model: The Serial Peripheral Interface (SPI) bus is, basically, a relatively simple synchronous serial interface for connecting low-speed external devices using quite minimal number of wires. The SPI-bus is a 4-wire serial communications interface used by many microprocessor peripheral chips. It provides support for a low/medium bandwidth (1 Mega Baud) network connection amongst CPUs and other devices supporting the SPI.

The SPI is synchronous and fully duplex, i.e., it uses a clock signal to time bit transfers in blocks of 8 bits, and one wire handles transmitted data and another handles received data. SPI bus is a master/slave interface. Whenever two devices communicate, one is referred to as the master and the other as the slave device. The master drives the serial clock. When using SPI, data is simultaneously transmitted and received, making it a full-duplex protocol.

The SPI signals are named as follows: SCLK for Serial Clock, which is always driven by the master: MISO is Master-In Slave-Out data: MOSI is Master-Out Slave-In Data. In a typical application, the microcontroller's SCLK is connected to the converter's SCLK input, the MISO is connected to the converter's DOUT pin, and the MOSI pin is connected to the converter's DIN pin. In most of the serial communication protocols, such as SPI, a chip-select input is required to enable the IC. Using this chip-select signal it is possible to connect many ICs to the same SPI bus in parallel. If there is a Chip-Select (CS) signal in use, it can be driven by a spare microcontroller, general-purpose output. Every IC connected to bus needs it's own chip-select signal line. Thus, when 10 devices are on the bus, 10 chip-select lines, in addition to the shared clock and data lines, are needed to select the appropriate device.

2) I2C Interface Model: The I2C (Inter-Integrated Circuit) interface enables data communication on a two-wire bi-directional bus - serial data (SDA) and serial clock (SCL) - between a small number of devices (sensors, microcontroller, LCD display, etc.). The I2C interface supports a parallel interface which is compatible with most standard microcontrollers/microprocessors, data transfer rate up to 100 kbits/s (standard transmission mode), 7-bit addressing mode, start/stop/acknowledge generation and detection and bus state (busy) detection.

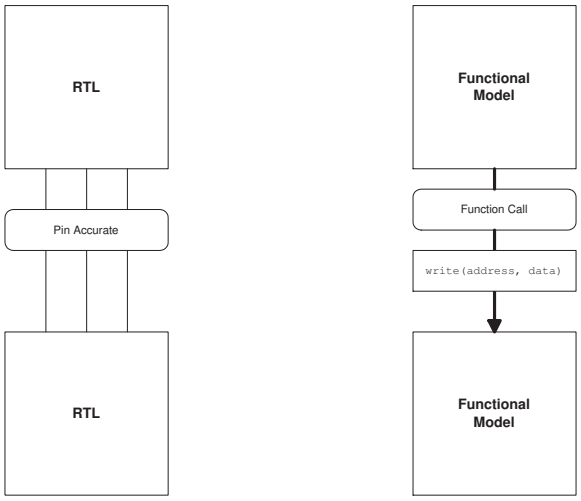


Fig. 2. Transaction-Level Model

The data frame for the standard mode is made of a start bit, a 7-bit address, a read/write bit, an acknowledge bit and a sequence of data bytes. Each data byte is followed by an acknowledge bit issued by the target device. A stop bit finalizes the transmission. Each bit is transmitted on SDA in conjunction with the SCL clock.

A start condition initiates data transfers which happens when a falling transition occurs on SDA while SCL is high. A stop condition ends data transfers which happens when a rising transition occurs on SDA while SDA is high. During the high state of SCL data is considered valid. Therefore, SDA signal must remain stable during this half period.

I2C allows multi-master communications and features an arbitration management protocol for such transmissions. The arbitration takes place on the SDA line, while the SCL line is in the high state. The control of the bus is granted to the master which transmits a low level while the others transmit a high level. A master which loses arbitration switches its data output stage to high impedance state.

The basic structure of the I2C bus controller comprises two distinct blocks: a digital block interfaced with a master (microprocessor core) manages the I2C protocol timing and control of specific sequences while an analog block ensures access to the external bus [8]–[10].

The digital block manages the acknowledge generation/detection depending on the mode of operation (transmit/receive). A shift register either serializes data to be sent to the bus line in transmit mode or collects information from the bus in receive mode. It also sets the transmission frequency by dividing the system clock with a user-defined constant.

The architecture of the digital block is divided into three blocks:

1) A processor interface handles all data transfers between the master and the I2C bus controller and interprets all the requests from the master (such as read data, write

data, bus controller configuration, etc.). It is built around a FIFO which stores the successive requests coming from the microprocessor bus (AVR bus, AMBA, VCI, etc.). When the bus controller has finished a transaction on the I2C bus and if a request is stored in the FIFO, the FIFO is read and the processor interface extracts the information needed by the sequencer (type of operation, address, data, etc.) to perform new communication. The processor interface also includes an interrupt line connected to the master so that it can read a data received from the I2C bus.

2) The core of the bus controller is the sequencer module which translates all the requests from the master into a detailed sequence respecting the I2C protocol such as frame generation (start/stop bits, address transmission), byte transmission or reception, etc.). It is composed of a set of finite state machines.
3) A signal generator module manages or drives the SCL and SDA bus lines according to the sequencer commands.

C. Radio and Sensor Channel Modeling with SystemC Hierarchical Channels

1) Radio Channel Model: Radio Irregularity is a common phenomenon with non-negligible effects in wireless sensor networks because it results in irregularity in radio range and variations in packet loss in different directions and is considered to have an impact on MAC, routing, localization and topology control protocols [11].

Several empirical studies have revealed that the radio range varies significantly in different directions and the percentage of asymmetrical links in a wireless sensor network system varies dependent upon the distance between the sensor nodes. Although the impact of radio irregularity on the protocol performance of a wireless sensor network can be investigated empirically, only few research groups have actually pursued this direction because of two main reasons: first, the complexity and cost of performance evaluations on an empirical system escalate when the nodes in a wireless sensor network scale up to thousands; second, repeatable results of radio performance are extremely hard to obtain from uncontrolled environments, hence, leading to difficulties in system tuning and performance evaluation.

The spherical radio footprints assumed by most existing simulation models do not approximate real radio properties well enough and, hence, lead to an inaccurate estimation of the application performance.

In general, radio irregularity is caused by the anisotropic properties of the propagation media and the heterogeneous properties of the physical radio devices.

While RF signal propagation models such as fading and path loss are not part of the radio model, they control the input given to the radio model and have great impact on their performance.

Fading: is a variation of signal power at receivers, caused by the node mobility that creates varying path conditions from the transmitters. The AWGN (Additive White Gaussian Noise) radio channel model is an idealistic channel condition where no signal fading occurs. The fading models with Rayleigh or Ricean distributions are commonly used to model wireless sensor network environments. The fading model with Rayleigh distribution is meant for highly mobile conditions with NLOS (No Line Of Sight) path between the communicating nodes, while the fading model with Ricean distribution accounts for the LOS (Line Of Sight) path between the communicating nodes. The signal power from the LOS path with respect to the signal power from NLOS paths can be controlled by a parameter called the Ricean K factor.

Path Loss: When an electromagnetic signal propagates within a medium, it may be reflected, diffracted and scattered. These effects have two important consequences on the signal strength. First, the signal decays exponentially with respect to distance. Second, for a given distance, d, the signal strength is random and log-normally, distributed about the mean distance-dependent value. The variance in the signal path loss is one of the major causes of radio irregularity.

Reflection occurs when an electromagnetic signal encounters an object, such as a building, that is greater than the wavelength of the signal. Diffraction occurs when the signal encounters an irregular surface such as a stone with sharp edges. Scattering occurs when the medium through which the electromagnetic wave propagates contains a large number of objects smaller than the wavelength of the signal. The properties of the communication medium are normally different in different directions. Consequently, radio propagation exhibits anisotropic patterns in most environments.

Path Loss defines the average signal power loss of a path on the terrain. The free-space path loss model is used as a basic reference model and is also considered to be an idealized propagation model. With this path loss model, even nodes far from the transmitter can receive packets, which can result in fewer hops to reach the final destination in wireless sensor networks. Therefore, simulation results with the free-space path loss model tend to be better than with other path loss models. However, as signal propagation with little power loss may cause stronger interference for concurrent transmissions, it does not necessarily yield the best performance under all scenarios. The two-ray path loss model is suited for LOS microcell channels in urban environments, and its use for wireless sensor networks can be justified by the environmental similarities (low transmit power and low antenna height).

In isotropic radio propagation models, the received signal strength is, usually, represented with the following formula:

$$\begin{aligned}\text{received signal strength} = {} & \text{sending power} + \text{fading} \\ & - \text{path loss}\end{aligned}$$

The sending power of a wireless sensor node is determined by the battery status and the type of the radio transmitter, power amplifier and the antenna. The path loss determines the signal's energy loss as it travels to the receiver. Many models are used to estimate the path loss, such as the free-space propagation model, the two-ray model, etc. However, all these models are isotropic, meaning that the signal attenuates exactly the same in all directions which do no hold well in practice. Therefore, the following formula is more accurate:

$$\text{received signal strength} = \text{sending power} + \text{fading} - \text{anisotropic path loss}$$

where, $\text{anisotropic path loss} = K_i \times \text{isotropic path loss}$
K_i is a coefficient to represent the difference in path loss in different directions.

Radio Interference Model: The computation of interference and noise at a receiver is a critical factor in wireless sensor network communication modeling, as this computation becomes the basis of SINR (Signal to Interference and Noise Ratio) or SNR (Signal to Noise Ratio) that has a strong correlation with FER (Frame Error Rate) on the radio channel. The power of interference and noise are calculated as the sum of all signals on the radio channel other than the one being received by the radio plus the thermal (receiver) noise. The resulting power is used as the base of SNR, which determines the probability of successful signal reception for a given frame. For a given SNR value, two signal reception models are commonly used in wireless network simulators: SNR threshold based and BER based models. The SNR threshold based model uses the SNR value directly by comparing it with an SNR threshold (SNRT), and accepts only signals whose SNR values have been above SNRT at any time during the reception. The BER based model probabilistically decides whether or not each frame is received successfully based on the frame length and the BER (Bit Error Rate) deduced by SNR and the modulation scheme used at the transceiver. As the model evaluates each segment of frame with a BER value every time the interference power changes, it is considered to be more realistic and accurate than the SNR threshold based model. However, the SNR threshold based model requires less computational cost and can be a good abstraction if each frame length is long.

This regards the model for radio interference in a wireless network simulation. Given the different ways in which each specific simulator may compute radio interference, it is important to know exactly what model drives this computation because this model has a substantial impact in determining the accuracy of the simulation's results.

The assessment of the strength of interference on a wireless node, however, comes at a high price in terms of computation. The total amount of interference on a node is the summation of all signals that can be picked up at its location which come from a source other than the sender of information. When the number of nodes in a wireless network model grows, not only does the number of terms in this summation grow fast, but also does the number of times the summation has to be computed. Clearly, without any measure to restrain the increase in the complexity of these computations, the scalability of the simulator can be severely impaired.

A common solution to reduce the computational complexity of interference calculations in wireless networking models is to limit the propagation range of interfering signals. In practice, this amounts to defining a cutoff value for radio signal propagation. The basic idea is that since interference is computed as the summation of all the "other" signals in a channel, sufficiently small terms in this summation could be discarded without substantially compromising the accuracy of the calculation. The crucial question here lies in determining how faint a signal should be so that it can be discarded from an interference computation without inducing substantial errors.

If a simulator should offer a cutoff parameter in the description of the experimental scenario, one should understand what consequences a chosen value brings.

This parameter can be interpreted in two different ways. It can be read as the maximum distance between transmitter and receiver that guarantees that the received signal is intelligible. Alternatively, cutoff can be defined as the highest attenuation (or path loss) that a signal may suffer and still be received (measured in decibels). We have taken the latter approach and require the user to enter this value in the configuration of the simulation scenario. Using a function provided by the underlying radio propagation model, the simulator converts this attenuation value to a distance value. Since different radio propagation models determine very different attenuations for the same transmitter-receiver separation, we believe this is the most general and practical solution. Note that the importance of a cutoff parameter extends beyond just determining the complexity of the interference computation. This parameter is used in the construction of a connectivity graph for the network, which determines what radio links exist between nodes. When a node sends out a radio frame the connectivity graph is inspected to that the simulator knows to what other nodes deliver the information. If the network nodes are mobile, this connectivity graph is updated periodically.

2) Sensor Channel Model: A typical wireless sensor node is equipped with one or more sensors, e.g., acceleration (seismic), acoustic, heat (temperature), pressure, etc. Each of these sensor types senses some physical phenomenon that propagates either through wave mechanics or diffusion. The former follows the inverse distance power law and suffers from fading, path-loss, multipath distortion, etc. (e.g., seismic waves, sound waves). The latter can be defined as the property of movement of species across a gradient from region of low concentration to high (e.g., heat, pressure). Both of these propagation models (channels) are the building blocks upon which any specific sensor channel can be defined. The wave propagation channel can be implemented similar to the radio propagation models. For diffusion channel, Fick's law can be used for temperature gradient Fourier's law for concentration gradient, etc.

IV. Conclusions

We have shown how to link the top-most, system-level and the lower-most, cycle-accurate platform models for wireless sensor networks through a bridging model which ties together the SystemC-based abstract RTOS model with the ISS to model SW and HW as well as models peripheral communications using submodels which are conceptually based on the transaction-level modeling theory and can be implemented in SystemC using the SystemC TLM library and hierarchical channels.

Acknowledgements

The work presented in this paper has been funded by the Danish National Research Council (STVF) project titled

”Hogthrob – Networked On-a-Chip Nodes for Sow Monitoring” (STVF 2059-03-0027).

REFERENCES

[1] Jerome Chevalier and Olivier Benny and Mathieu Rondonneau and Guy Bois and El Mostapha Aboulhamid and Francois-Raymond Boyer, “SPACE: A Hardware/Software SystemC Modeling Platform Including an RTOS,” *Languages for System Specification, Springer*, pp. 91–104, May 2004.

[2] Kathy Dang Nguyen and Zhenxin Sun and P. S. Thiagarajan, “Model-Driven SoC Design Via Executable UML to SystemC,” in *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)*, 2004, pp. 459–468.

[3] Jayaram Bhasker, *A SystemC Primer*. ISBN: 0-9650391-8-8: Star Galaxy Publishing, 2002, pp 268.

[4] David C. Black and Jack Donovan, *SystemC: From the Ground Up*. ISBN: 1-4020-7988-3: Kluwer Academic Publishers, 2004, pp 244.

[5] Thorsten Grotker and Stan Liao and Grant Martin and Stuart Swan, *System Design with SystemC*. ISBN: 1-4020-7072-1: Kluwer Academic Publishers, 2002, pp 217.

[6] Frank Ghennassia, *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. ISBN: 0-387-26232-6: Springer, 2005, pp 271.

[7] Kashif Virk and Knud Hansen and Jan Madsen, “System-Level Modeling of Wireless Integrated Sensor Networks,” in *Proceedings of the IEEE International Symposium on System-on-Chip (SoC 2005)*, November 2005.

[8] Ahmed Habbani and Olivier Romain and Patrick Garda, “MIS and I2C Bus,” *International Journal of Computer Sciences and Engineering Systems*, vol. 1, no. 2, pp. 111–117, April 2007.

[9] Mohammad Alassir and Julien Denoulet and Olivier Romain and Patrick Garda, “Modeling I2C Communication between SoCs with SystemC-AMS,” in *Proceedings of the 2007 International Symposium on Industrial Electronics*, June, 2007, pp. 1412–1417.

[10] M. Alassir and J. Denoulet and O. Romain and P. Garda, “A SystemC AMS Model of an I2C Bus Controller,” in *Proceedings of the 2006 International Conference on Design and Test of Integrated Systems in Nanoscale Technology*, September, 2006, pp. 154–158.

[11] Gang Zhou and Tian He and Sudha Krishnamurthy and John A. Stankovic, “Models and and Solutions for Radio Irregularity in Wireless Sensor Networks,” *ACM Transactions on Sensor Networks (TOSN)*, vol. 2, no. 2, pp. 221–262, May 2006.

Chapter 8

Design of a Development Platform for HW/SW Codesign of Wireless Integrated Sensor Nodes

Kashif Virk, Martin Leopold, Andreas Vad Lorentzen, Martin Hansen, Phillipe Bonnet and Jan Madsen. Design of a Development Platform for HW/SW Codesign of Wireless Integrated Sensor Nodes. *Proceedings of the IEEE EuroMicro Conference on Digital System Design* (DSD'05), August 2005. Pages: 254-260. Published.

Design of A Development Platform for HW/SW Codesign of Wireless Integrated Sensor Nodes

Kashif Virk,
Jan Madsen,
Andreas Vad Lorentzen
System-on-Chip Group
Technical University of Denmark
email: {virk, jan}@imm.dtu.dk

Martin Leopold,
Phillipe Bonnet
Distributed Systems Laboratory
Institute of Computing Science
Copenhagen University, Denmark
email: {leopold, bonnet}@diku.dk

Martin Hansen
Valby Design Center
IO Technologies A/S
email: martin@iotech.dk

***Abstract*—Wireless integrated sensor networks are a new class of embedded computer systems which have been made possible mainly by the recent advances in the micro and the nano technology. In order to efficiently utilize the limited resources available on a sensor node, we need to optimize its key design parameters which is only possible by making system-level design decisions about its hardware and software (operating system and applications) architecture. In this paper, we present the design of a sensor node development platform in relation to an application of wireless integrated sensor networks for sow monitoring. We also discuss the related hardware/software codesign tradeoffs.**

I. Introduction

Wireless integrated sensor networks are a class of networked embedded systems that combine sensing, computation, and communication in an inexpensive and very small form factor device with limited energy. They are meant to act as a bridge between the physical and the virtual worlds. Apart from myriads of applications being proposed for them, their low cost and size, ease of deployment, and autonomous operation make them a viable and non-intrusive solution for livestock monitoring applications.

In the Hogthrob [1] project, we are aiming to develop a sensor network infrastructure for sow monitoring. A part of the project consists of developing sensor nodes that can be tagged onto the sows (in replacement of the RFID tags they wear today). Such sensor nodes must be low-cost (costing no more than a couple of Euros), small-sized (small enough, when packaged, to be worn as an ear tag) and low-energy (a few months' autonomy is a minimum). In order to conform to the above-mentioned requirements, we have decided to design a sensor node on a chip.

Today's Danish farms for pig production are using RFID tags for sow identification and controlling their food consumption. However, these tags have proven to be quite impractical to locate sows in large pens. Moreover, they are not flexible enough to be useful in contexts other than controlling the food consumption. For example, the pig farmers have to manually monitor the key aspects of a sow's lifecycle such as the onset of estrus[1] or farrowing - the phenomena that have a profound effect on pig production.

[1]Estrus or Heat Period is the period when a sow can be bred and it lasts for a short time only. If a sow is not bred during its first estrus, it is considered unproductive from the commercial point of view since it normally returns to estrus about 3 weeks later and needs to be fed and housed meanwhile.

Numerous sensor network research projects have designed sensor nodes with microprocessors from Atmel, Texas Instruments, Intel, etc. for similar purposes, notably [2], [3]. However, none of the sensor node architectures, reported so far in the literature, approaches the sensor node design from a hardware/software codesign perspective (except in [4], but to a limited extent). We have designed a sensor node development platform in order to explore the design-space both in terms of hardware and software and to end up with a complete sensor node implemented on a single chip. Hence, a key component of our sensor node development platform is an FPGA which has enabled us to explore various hardware/software tradeoffs.

An important design consideration while designing our sensor node development platform, called the Hogthrob platform, has been to reduce the overall cost of prototyping by using COTS (Common Off-the-Shelf) components. Another consideration has been to have the capability to experiment with various combinations of sensors, radio transceivers (e.g., Bluetooth, Zigbee, Wi-Fi, UWB, etc.), and microprocessors (e.g., Atmel, Intel, ARM, Texas Instruments, Microchip, etc.) to select the optimal combination. To achieve these objectives, we have adapted a modular design strategy so that we can swap sensors and radio transceivers with the ones resulting in more efficient energy and system performance. For trying different microprocessors and/or to perform hardware acceleration, we needed some form of reconfigurable logic on the sensor node development platform so that we can configure the sensor node with various mircoprocessor cores. Of course, low power, small form factor, and robust packaging were the necessary features as well because the sensor nodes have to be mounted on sows.

The Hogthrob platform has also been designed with a view to explore the tradeoffs of implementing application functionality either in software (on the embedded processor) or hardware (on the reconfigurable logic), without being constrained by the initial design choices as was the case in [5]. As an initial design step, all the application functionality has been placed on the embedded processor and is gradually being moved to the FPGA. At the the current stage of software development, the radio transceiver and other peripherals are being controlled by the software running on the embedded processor but, eventually, the embedded processor will only initialize the FPGA and function as an external timer and an A/D converter for the FPGA.

II. Sensor Node Hardware Architecture

The Hogthrob platform architecture consists of four closely-interacting subsystems (please refer to Figure 2). These subsystems are: the sensing subsystem, the computing subsystem, the communication subsystem, and the power-supply subsystem. The platform has been designed using a modular design approach and comprises one mother board (8.5cm x 7cm) which comprises the computing and the power-supply subsystems, one daughter board for the communication subsystem (4cm x 5cm) and another daughter board for the sensing subsystem. The mother board has been further divided into the analog and the digital sections with the analog section mostly occupied by the power-supply subsystem and the computing subsystem comprising the digital section.

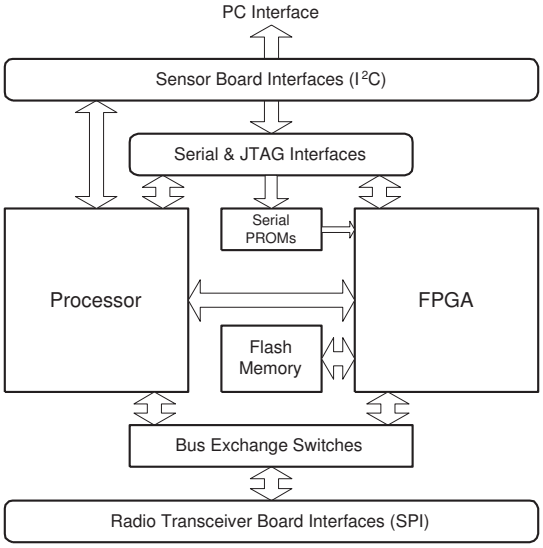


Fig. 1. *Hogthrob Sensor Node Development Platform Interfaces*

A. Sensing Subsystem

The sensing subsystem can support an assortment of analog and digital sensing devices. As the Hogthrob project is still in its infacy, the requirements for monitoring the various parameters that can be associated with the phenomenon of the onset of estrus in sows might change. However, preliminary studies have indicated a direct correlation between the movement of a sow and the onset of estrus [6]. Therefore, at present, we are looking into the use of a MEMS-based, three-axis accelerometer with a digital output (possibly, the most recently-released, LIS3L02D, single-chip device from ST Microelectronics [7]). In future, we might have to use temperature and accoustic sensors which can have analog outputs. The analog outputs from these sensors can be processed by the 10-bit A/D converter available on the ATMega128L either directly or, if necessary, one analog input can be routed via an on-chip comparator for signal conditioning. The A/D converter supports 8 analog input channels. The ATMega128L also supports an I^2C interface which is commonly available on most of the sensors.

B. Computing Subsystem

The computing subsystem is centered around the Atmel ATMega128L microcontroller running at a clock frequency of 8 MHz at 3.0V and the Xilinx Spartan3 series XC3S400 FPGA (please see Table I) running at the clock frequencies of 48MHz and 4MHz at 2.5V for the peripherals and 1.2V for the core. The primary function of the computing subsystem is to execute the sow monitoring application and to coordinate the functions of the sensor node. The operating system running on the sensor nodes forms the core of the software running on the computing subsystem which is responsible for the task scheduling operations and resource management. We are presently running the TinyOS which is an event-based embedded operating system developed at the University of California, Berkeley [8] (please see Section 4).

There are a number of interfaces supported by the ATMega128L which are also supported by the mother board (please refer to Figure 1). These interfaces include the two-wire (I^2C) interface for the sensing subsystem, the three-wire (SPI) interface for the communication subsystem, the JTAG interface for in-system programmability and debugging, and the serial (RS-232) interface for interaction with the PC.

The Spartan3-series FPGA has been included to act either as a hardware accelerator for the ATMega128L or it can be configured with a stand-alone microprocessor core working independently of the on-board ATMega128L. This will allow us to experiment with various microprocessor/microcontroller cores without redesigning the mother board.

The FPGA supports the same interfaces as the ATMega128L [9]. The access of either the ATMega128L or the FPGA to the radio transceiver is controlled by the bus exchange switches. The interface between the ATMega128L and the FPGA consists of parallel multiplexed address and data buses which can be demultiplexed by implementing an address latch in the FPGA and using the ALE (Address Latch Enable) signal. The FPGA uses an in-system programmable Flash memory (4M x 16 bits). There are two serial configuration PROM's provided with the FPGA which are controlled by the ATMega128L and the configuration data can be downloaded to them through the JTAG port. A number of LED's and push-buttons have been provided at the mother board for easy test and debugging.

C. Communication Subsystem

The communication subsystem manages the data transfer and signaling (beaconing) between the sensor nodes. It includes the network protocols and the radio transceiver. The radio transceiver is a Nordic VLSI nRF2401 (please see Table I) that can deliver a maximum data rate of 1 Mbps. It consists of a fully-integrated frequency synthesizer which can generate a frequency range of 2.4-2.5 GHz in the ISM[2] band, a power amplifier, a crystal oscillator (it uses a 16MHz external crystal), and a GFSK[3] modulator. The output power and the frequency channels of the radio transceiver are fully programmable through the 3-wire (SPI) interface. The antenna

[2]ISM stands for Industrial, Scientific, and Medical
[3]GFSK stands for Gaussian-filtered Frequency-Shift Keying

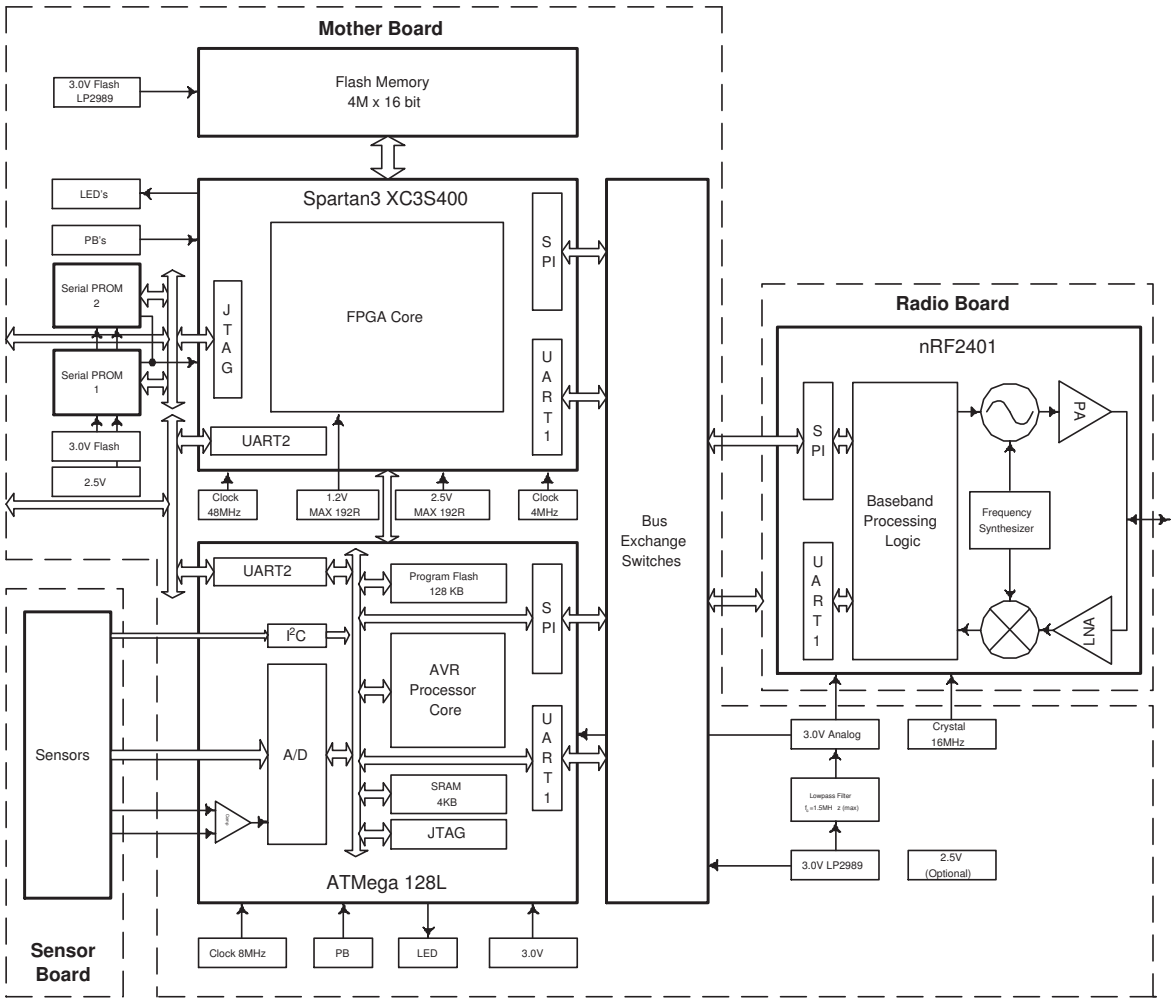


Fig. 2. *Hogthrob Sensor Node Development Platform Architecture*

used for radio transmission and reception is an omnidirectional stub antenna. The transmission range of the radio transceiver has been measured to be 80 meters under ideal conditions.

D. Power Supply Subsystem

The power-supply subsystem comprises 3 or 4 AAA-sized batteries and a collection of DC-DC converters to service the entire sensor node. The voltages generated by the DC-DC converters are: 3.0V, 3.0V-Flash, 2.5V, and 1.2V. The 3.0V supplied to the radio transceiver is filtered through a low-pass passive LC filter (f_c=1.5MHz) to generate 3.0V-Analog which is also used by the ATMega128L for its analog I/O. The two serial configuration PROM's for the FPGA use two supplies: 3.0V-Flash and 2.5V. An optional 2.5V is also reserved for the radio transceiver for future use.

ATMega128L Resources		**Spartan3 Resources**		**nRF2401 Characteristics**	
EEPROM	4KB	*Gates*	400K	*Carrier Freq.*	2.4 GHz
RAM	4KB	*CLB Size*	32x28	*Modulation*	GFSK
I/O's	53	*Slices*	3,584	*Data Rate*	0-1 Mbps
8-Bit Timers	2	*Logic Cells*	8,064	*Sensitivity*	-90 dBm
16-Bit Timers	2	*CLB FF's*	7,168	*Voltage*	1.9-3.6V
10-Bit ADC Chan.	8	*Dist. RAM*	56K Bits	*Current: TX*	10.5 mA
SPI Interface	1	*RAM Blocks*	16	*Current: RX*	18 mA
I^2C *Interface*	1	*Block RAM*	288K Bits	*Current: PD*	1 μA
JTAG Interface	1	*Config. PROM*	1.7M Bits	*Channels*	25
UART Interface	2	*Max. I/O*	264	*Sec. Mod.*	FH-SS

TABLE I
Salient Features of the Hogthrob Platform

III. Sensor Node Software

The sensor node is limited in a number of ways, memory, computational power, etc. However, the most limited resource is the energy. The energy performance of a sensor node is greatly influenced by the software running on it. The sensor node control software (or the operating system) has to be designed to efficiently utilize the limited resources and, especially, the power-conserving features of the sensor node platform and to incur low computation and communication overhead. Furthermore, the application software has to take advantage of the spatial and temporal characteristics of the targeted application; it must *model* the application as realis-

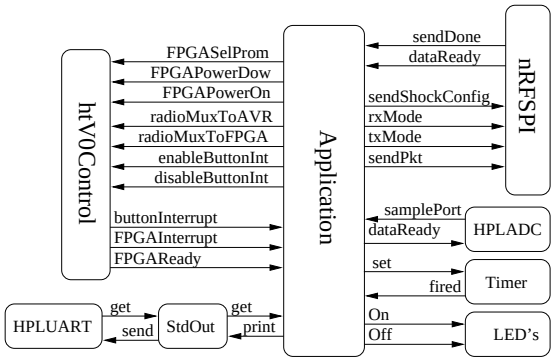


Fig. 3. *TinyOS Components*

tically as possible. In the Hogthrob project, the focus of our application is the accurate and reliable detection of the estrus of sows.

As mentioned in Section 3, we are running TinyOS – an event-based embedded operating system, on the Hogthrob platform. The extremely modular and flexible design of TinyOS is very well-suited for exploring the boundary between hardware and software [10]. TinyOS is a programming environment rather than an operating system in the traditional sense and is closely tied to the nesC language which is an extension of the C-language [11]. The TinyOS programs comprise a number of *components* interconnected by *interfaces*. A component implements an interface, and can serve either as a software module or as a wrapper for a hardware block. All the components of a TinyOS program are compiled into a single static image which is uploaded to the target platform. Compiling a static image allows optimizations that are otherwise troublesome in a traditional operating system, such as whole program analysis, compile-time data race analysis and more detailed dead-code detection.TinyOS was originally developed for the Mica sensor node platform [12], and has been, sucessfully ported to the Hogthrob platform.

A. Porting TinyOS to the Hogthrob Platform

The core of TinyOS has no platform-dependent components [13], therefore, the process of porting TinyOS to the Hogthrob platform has been fairly straightforward. In addition, we have implemented two TinyOS components, `nRFSPI` and `htV0Control`, to access the radio transceiver, the FPGA, the bus-exchange switches, and the push-buttons (please see Figure 3):

nRFSPI: The Hogthrob platform software is a work in progress and is currently limited to simple connectionless unreliable communication. Although the nRF2401 radio transceiver implements major parts of the physical access protocol, it does not handle collisions and retransmissions. The nRFSPI component functions as a wrapper for this functionality and provides a *packet-level* interface to the *byte-level* SPI peripheral of the ATMega.

htV0Control: This component implements the methods and the events closely tied to the Hogthrob platform – selecting the PROM for the FPGA configuration, booting the FPGA, setting up the FPGA communication, setting the bus-exchange switches and the push-button interrupts.

As the configuration of the Hogthrob platform is quite similar to the Mica platform [12], we can use most of the other hardware components of TinyOS (e.g., the A/D Converter, UART's, LED's, Timer's) with minor modifications.

B. The Sow Monitoring Application

In collaboration with KVL[4] (which is our partner university in the Hogthrob project), we are studying the behavior of a herd of sows before and during their estrus. Thus far, we have identified the following characteristic behavioral features which are relevant to the application software running on the sensor nodes.

1) During the night, the sows rest (sleep) for long periods (4-8 hours at a stretch)
2) The heat period occurs infrequently but regularly suggesting different *duty cycles* of operation – not in heat period (low sample rate); might be in heat period (high sample rate).

With these observations, it seems appropriate to power down a sensor node while a sow is sleeping and to duty-cycle the sensor nodes according to the sow activity. Our initial approach is to formulate a model (based on the Markov Model or the Finite State Automata) and associate a duty cycle with each state (e.g., sleeping - one sample per hour, active - one sample per minute, close to a boar - 4 samples per second).

As mentioned earlier, it has been shown that there is a correlation between the movement of a sow and the onset of its heat period [6]. We are now conducting experiments to get initial time series characterizing the movement of sows using accelerometers. Along with KVL, we will develop an application model based on these time series.

IV. Hardware-Software Codesign of the Sensor Node

In order to explore various options available for designing the computing subsystem of our sensor node SoC, we have designed and implemented a custom microprocessor core on the FPGA. The design of low-power microprocessors for portable systems is an ongoing research subject [14], [15], however, our focus in this project is not to advance the field of low-power microprocessor design. Therefore, we have made our design decisions by selecting from relatively well-understood options.

As mentioned above, for our sensor network application, the microprocessor will spend most of its time in sleep mode – not actively executing instructions. Thus, our main design focus has to be on reducing startup times and providing the right low-power states and hardware accelerators and not on executing the program instructions efficiently.

Furthermore, a key issue is the availability of design automation tool chain. Having a suite of design automation tools available while developing an actual application is a must. As an example, consider the Freescale evaluation board [16] based on the Motorola HCS08 microcontroller, which is not supported by the GNU GCC compiler. Porting TinyOS to this

[4]The Royal Veterinary and Agricultural University, Copenhagen, Denmark

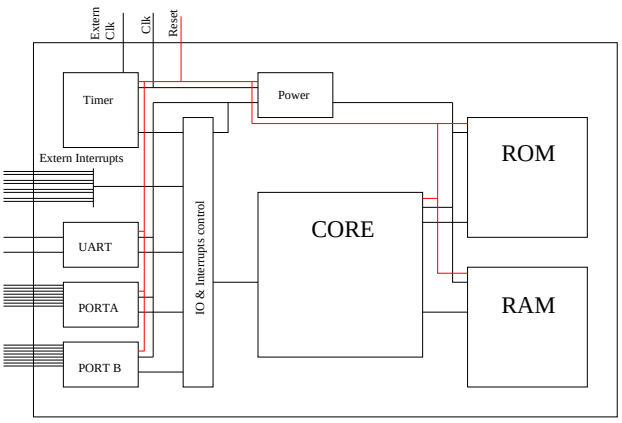


Fig. 4. *Nimbus Microprocessor Core*

platform not only involves rewriting the hardware drivers, but also the tool chain differences. These considerations have led us to implement an AVR-instruction set compatible processor, which we call Nimbus after a classic Danish motorcycle.

A. Nimbus Processor Core

The Nimbus processor core (please see Figure 4) implements a subset of the Atmel ATMega103 functionality with no hardware multiply unit. It is based on the AVR processor core from OpenCores [5] which has been debugged and modified to support different power modes. The serial flash is used for program storage. However, for simplicity, the programs are stored in the SRAM memory blocks of the FPGA while testing.

To get a feel of how the Nimbus processor core performs, we have simulated the Nimbus core using the Synopsys Power Compiler and compared it with the Atmel ATMega128L.

For comparing the two processor cores, we have written a few benchmarks and have run them in the simulated environment of the Nimbus processor core and on the Atmel ATMega128L on a BTNode2 [17]. Each of these benchmarks exercise different features of each processor core. To allow us to measure just the power consumption of the processor core and no additional component, we have slightly modified[6] the BTNode2 board. The Atmel ATMega128L, on the BTNode2, runs at 7.35 MHz and is powered with a 3.3 V power supply while the Nimbus processor core is simulated to be running at 7.0 MHz with a 1.32 V power supply.

In order to explore and compare the architectural advantages of the Nimbus processor core over the Atmel ATMega1281, the two technologies have to be the same. However, the actual technology of the Atmel ATMega128l is not published. Therefore, we have made an assumption that it is manufactured using a 0.25 micron technology.

We summarize the comparison results in Table II. Although the simulated operating voltage of the Nimbus core is lower than that of the Atmel ATMega128L, but even with this simple comparison, with this relatively simple implementation

[5]www.opencores.org

[6]on the BTNode2 board, there is a 0 Ohm resistor that can easily be replaced by two wires, allowing the use of an ammeter

Benchmark	Nimbus	ATMega	Description
nop	2.26 mW	47.5 mW	tight loop of no operation instr.
idle	1.00 μW	17.0 mW	idle mode of the ATMega
power-save	1.22 μW	38.6 μW	power-save mode of the ATMega
power-down	0.59 μW	39.0 μW	power-down mode of the ATMega
add	1.38 mW	30.1 mW	tight loop of add instr. stored in registers
add-mem	1.90 mW	31.9 mW	tight loop of add instr. stored in memory
hamming	1.76 mW	32.3 mW	Hamming encoding and decoding

TABLE II
Comparison of the Nimbus Microprocessor Core with the ATMega128L Microprocessor

of the Nimbus processor core, it is evident that lowering the power consumption of the microprocessor in the computing subsystem is possible.

The results are summarized in Table II. It is clear from the results that, using this comparison, the Nimbus core outperforms the Atmel ATMega128l. While this is promising, our conclusion might be biased given the fact that we are using a lower operating voltage for the Nimbus core and if our assumption that the Atmel ATMega128L is manufactured using a 0.25 micron technology is wrong. The numbers are, obviously, not comparable in that case.

V. Work in Progress

To further explore the HW/SW codesign options and to enable platform tuning [18], we are busy capturing the SNAP [19] and BitSNAP [20] processor designs in GEZEL [21]. We are also exploring various sensor node platform modeling approaches in Java [22]–[24], GME [25], and SystemC [26].

The prototype sensor node is presently undergoing field trials and, once successful, we plan to build a second prototype which will be more compact and more customized. We will also compare the synchronous AVR core with its asynchronous counterpart. The ultimate goal is to shrink the sensor node to a single system-on-chip costing less than a couple of Euros. We are also planning to exploit a number of energy scavenging solutions being made available by the ongoing research in the fields of micro and nano technology [27], [28].

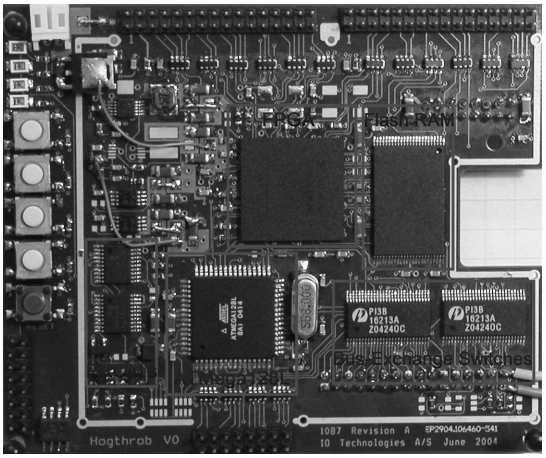


Fig. 5. *Motherboard of the Hogthrob Sensor Node Hardware*

ACKNOWLEDGEMENTS

The work presented in this paper has been funded by the Danish National Research Council (STVF) project titled "Hogthrob – Networked On-a-Chip Nodes for Sow Monitoring" (STVF 2059-03-0027).

REFERENCES

[1] The Hogthrob Project, "http://www.hogthrob.dk."

[2] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet," in *ASPLOS X*. ACM Press, 2002, pp. 96–107.

[3] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring," in *WSNA'02*. ACM Press, September 2002, pp. 88–97.

[4] Enz, C.C.; El-Hoiydi, A.; Decotignie, J.-D.; Peiris, V., "WiseNET: An Ultra Low-Power Wireless Sensor Network Solution," *IEEE Computer*, vol. 37, no. 8, pp. 62–70, August, 2004.

[5] M. Leopold, M. Dydensborg, and P. Bonnet, "Bluetooth and Sensor Networks: A Reality Check," in *Proceedings of the First International Conference on Embedded Networked Sensor Systems*, November 2003, pp. 103–113.

[6] R. Geers, S. Janssens, J. Spoorenberg, V. Goedseels, J. Noordhuizen, H. Ville, and J. Jourquin, "Automated Oesterus Detection of Sows With Sensors for Body Temperature and Physical Activity," in *Proceedings of ARBIP95, Kobe, Japan*, 1995.

[7] ST Microelectronics, Geneva, "Micro Electro-Mechanical Systems," http://www.st.com/mems.

[8] Univeristy of California, Berkeley, "TinyOS Community Forum," http://www.tinyos.net.

[9] Kashif Virk, "Testing FPGA Interfaces on Hogthrob Motherboard," Computer Science & Engineering, Informatics & Mathematical Modeling, Technical University of Denmark, Tech. Rep., December 2004.

[10] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in *SenSys'03*, November 2003.

[11] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems," in *Proceedings of Programming Language Design and Implementation (PLDI)*, June 2003.

[12] J. L. Hill and D. E. Culler, "MICA: A Wireless Platform for Deeply Embedded Networks," *IEEE Micro*, vol. 22, no. 6, pp. 12–24, November 2002.

[13] V. Handziski, J. Polastre, J.-H. Hauer, C. Sharp, A. Wolisz, and D. Culler, "Flexible Hardware Abstraction for Wireless Sensor Networks," in *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN 2005)*, 31 January - 2 February 2005.

[14] T. D. Burd and R. W. Brodersen, "Energy-Efficient CMOS Microprocessor Design," in *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS'95)*, 1995.

[15] ——, "Processor Design for Portable Systems," *Journal of VLSI Signal Processing Systems - Special issue on Technologies for Wireless Computing*, vol. 13, pp. 203–221, 1996.

[16] Freescale Semiconductor, Inc., "MC13192 Evaluation Board Reference Manual, Revision 1.0," http://www.freescale.com, September 2004.

[17] J. Beutel and O. Kasten and M. Ringwald, "Prototyping Wireless Sensor Networks with BTNodes," in *Proceedings of First European Workshop on Wireless Sensor Networks (EWSN 2004)*, January 2004.

[18] Frank Vahid and Tony Givargis, "Platform Tuning for Embedded Systems Design," *IEEE Computer*, vol. 34, no. 2, pp. 112–114, 2001.

[19] Clinton Kelly IV and Virantha N. Ekanayake and Rajit Manohar, "SNAP: A Sensor-Network Asynchronous Processor," in *Proceedings of 9th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2003), Vancouver, Canada*. IEEE Computer Society, 12-16 May 2003, pp. 24–35.

[20] Virantha N. Ekanayake and Clinton Kelly IV and Rajit Manohar, "BitSNAP: Dynamic Significance Compression for a Low-Energy Sensor Network Asynchronous Processor," in *Proceedings of 11th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2005), New York, USA*. IEEE Computer Society, 14-16 March 2005, pp. 144–154.

[21] GEZEL, "http://www.ucla.edu/ schaum/gezel."

[22] Ben Titzer and Daniel K. Lee and Jens Palsberg, "Avrora: Scalable Sensor Network Simulation with Precise Timing," in *Proceedings of Fourth International Conference on Information Processing in Sensor Networks, Joint IPSN 2005 Main Track and Special Track on Platform Tools and Design Methods for Networked Embedded Sensors (SPOTS), Sunset Village, UCLA, Los Angeles, USA*, April 25-27, 2005.

[23] Ben Titzer and Jens Palsberg, "Nonintrusive Precision Instrumentation of Microcontroller Software," in *Proceedings of Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2005), Chicago, Illinois, USA*, June 15-17, 2005.

[24] O. Landsiedel, K. Wehrle, B. L. Titzer, and J. Palsberg, "Enabling detailed modeling and analysis of sensor networks," *Praxis der Informationsverarbeitung und Kommunikation (PIK Journal) - Special Issue on Sensor Networks*, 2005, in press.

[25] GME, "http://www.isis.vanderbilt.edu/Projects/gme."

[26] SystemC Workgroup, "http://www.systemc.org."

[27] S. Roundy, P. K. Wright, and J. Rabaey, "A Study of Low-Level Vibrations as a Power Source for Wireless Sensor Nodes," *Computer Communications*, vol. 26, pp. 1131–1144, 2003.

[28] A. Kansal and M. B. Srivastava, "An Environmental Energy Harvesting Framework for Sensor Networks," in *Proceedings of the 2003 International Symposium on Low-Power Electronics and Design*, 2003, pp. 481–486.

Chapter 9

Functional Testing of Wireless Sensor Node Designs

Kashif Virk and Jan Madsen. Functional Testing of Wireless Sensor Node Designs. *Proceedings of the IEEE International Conference on Innovations in Information Technology* (Innovations'07), November 2007. Pages: 123-127. Published.

Functional Testing of Wireless Sensor Node Designs

Kashif Virk and Jan Madsen
System-on-Chip Group
Computer Science & Engineering Section
Department of Informatics & Mathematical Modeling
Technical University of Denmark, Lyngby 2800, Denmark
email: {virk, jan}@imm.dtu.dk

***Abstract*—Wireless sensor networks are networked embedded computer systems with stringent power, performance, cost and form-factor requirements along with numerous other constraints related to their pervasiveness and ubiquitousness. Therefore, only a systematic design methdology coupled with an efficient test approach can enable their conformance to design and deployment specifications. We discuss off-line hierarchical functional testing of complete wireless sensor nodes containing configurable logic through a combination of FPGA-based board test and Software-Based Self-Test (SBST) techniques. The proposed functional test methodology has been applied to a COTS-based sensor node development platform and can be applied, in general, for testing all types of wireless sensor node designs.**

I. Introduction

A number of, often conflicting, design, deployment and performance constraints increase the importance of functional testing in the context of wireless sensor node designs. In order to reduce the costs of low-volume wireless sensor node designs for research/proof-of-concept purposes, manufacturing test mechanisms like, AOI (Automated Optical Inspection), AXI (Automated X-ray Inspection), etc. cannot be employed, thus, reducing the possibility for the detection of process-induced faults. These test mechanisms become feasible only for certain minimum production quantities (which, usually, run in thousands) when their cost can be amortized over high production volumes. Other factors reducing the test coverage of wireless sensor node hardware are the form-factor limitations coupled with high pin-count, fine lead-pitch surface-mount (e.g., TSOP, TQFP, etc.) and area-array component packages (e.g., μBGA, CSP, Flip-Chip, etc.) which necessitate multi-layer PCB design with no provision for test node insertion, thus, circumventing even Flying Probe ICT (In-Circuit Test). Apart from being intrusive and costly, high-speed testers are increasingly becoming unable to match the component speeds [1]. Built-in Test mechanisms can be implemented using a Built-In Self Test (BIST) infrastructure incorporated into the wireless sensor node designs. However, hardware BIST is often not possible because dedicated test circuitry incurs a performance, area and energy overhead as described in [1] and is, in general, not preferable for low-cost, low-power wireless sensor nodes [2], which are often built with Commercial Off-The-Shelf (COTS) components including Systems-on-Chip [3] and all of the COTS components might not support BIST. Therefore, functional testing at system speed seems to be the only viable option available for testing wireless sensor node hardware to detect process-induced faults, as well as, design-related errors while keeping the costs to a minimum.

In this paper, we describe an at-speed, functional test methodology developed for the Hogthrob Project [4] which is hierarchical in the sense that it first performs a board-level test for testing the on-board bus interconnects (for shorts, continuity, signal integrity, etc.) and interfaces among the COTS components (for design-related errors, at-speed timing issues, etc.) by implementing a synthesizable Test Controller in the on-board reconfigurable logic which functions both as a test vector generator and a response analyzer and achieves a high fault coverage. The component-level testing is performed next on major system components (Flash memory, microcontroller, etc.). The main contributions of our work are a combination of various component-level (March Test, Software-Based Self-Test, etc.) and board-level (at-speed test, interconnect test, etc.) functional test strategies into a coherent, hierarchical test methodology aimed at testing the implementations of wireless sensor node architectures while taking into account all the constraints (functional, structural, etc.) outlined above and the exploitation of the on-board reconfigurable logic for implemeting a Test Controller which has a unique access to all the board-level and component-level interfaces.

Our functional test methodology, presented here, also augments the Software-Based Self Test (SBST) approach described in [5] for testing the embedded microcontroller and in [6] for testing the on-board Flash memory while extending, as well, the approach described in [7] (without any considerations for power efficiency because our objective has only been low-cost, off-line testing) by testing not only the COTS components comprising the sensor node system (with an FPGA, in addition) but by also testing the interconnections on the printed circuit boards comprising the wireless sensor node platform. It can be applied, in general, for the functional testing of low-volume wireless sensor node designs in a totally self-contained and non-intrusive manner.

The rest of this paper is organized as follows: a description of our system under test is given in Section II. Section III elaborates on the FPGA-based Test Controller for board-level testing of the peripheral interfaces as well as the Flash memory and Section IV discusses the software-based self test of the microcontroller. The conclusions are provided in Section V followed by a description of the work in progress in Section VI.

II. System Description

Designing wireless sensor nodes for Wireless Sensor Networks (WSN's) is an error-prone and, hence, an iterative pro-

cess because of the inherent intricacies of designing a wireless communication-oriented, mixed-signal, distributed embedded system. Therefore, it is important to follow a systematic design methodology coupled with an efficient test approach to satisfy all the design requirements for the target application. An important design consideration while designing our wireless sensor node development platform, called the Hogthrob platform, has been to reduce the overall cost of prototyping by using COTS components. The Hogthrob platform has also been designed with a view to explore the HW/SW tradeoffs, therefore, it also contains reconfigurable logic.

The Hogthrob platform architecture consists of four closely-interacting subsystems (please refer to Figure 1). These subsystems are: the sensing subsystem, the computing subsystem, the communication subsystem, and the power-supply subsystem. The platform has been designed using a modular design approach and comprises one mother board (8.5cm$\times$7cm) which comprises the computing and the power-supply subsystems, one daughter board for the communication subsystem (4cm$\times$5cm) and another daughter board for the sensing subsystem. The mother board has been further divided into the analog and the digital sections with the analog section mostly occupied by the power-supply subsystem and the computing subsystem comprising the digital section.

A. Sensing Subsystem

The sensing subsystem can support an assortment of analog and digital sensing devices. At present, we are looking into the use of a MEMS-based, three-axis accelerometer with a digital output. In future, we might have to use temperature and accoustic sensors which can have analog outputs. The analog outputs from these sensors can be processed by the 10-bit A/D converter available on the ATMega128L either directly or, if necessary, one analog input can be routed via an on-chip comparator for signal conditioning. The A/D converter supports 8 analog input channels. The ATMega128L also supports an I2C interface which is commonly available on most of the sensors.

B. Computing Subsystem

The computing subsystem is centered around the Atmel ATMega128L microcontroller running at a clock frequency of 8 MHz at 3.0V and the Xilinx Spartan3 series XC3S400 FPGA running at the clock frequencies of 48 MHz and 4 MHz at 2.5V for the peripherals and 1.2V for the core. The primary function of the computing subsystem is to execute the wireless sensor network application and to coordinate the functions of the sensor node. The operating system running on the sensor nodes forms the core of the software running on the computing subsystem which is responsible for the task scheduling operations and resource management. We are, presently, running a port of TinyOS to the Hogthrob platform which is an event-based embedded operating system developed at the University of California, Berkeley [8].

There are a number of interfaces supported by the ATMega128L which are also supported by the mother board (please refer to Figure 1). These interfaces include the two-wire (I2C) interface for the sensing subsystem, the three-wire (SPI) interface for the communication subsystem, the JTAG interface for in-system programmability and debugging, and the serial (RS-232) interface for interaction with the PC. The Spartan3-series FPGA has been included to act either as a hardware accelerator for the ATMega128L or it can be configured with a stand-alone microprocessor core working independently of the on-board ATMega128L. This will allow us to experiment with various microprocessor/microcontroller cores without redesigning the mother board.

The FPGA supports the same interfaces as the ATMega128L. The access of either the ATMega128L or the FPGA to the radio transceiver is controlled by the bus exchange switches. The interface between the ATMega128L and the FPGA consists of parallel multiplexed address and data buses which can be demultiplexed by implementing an address latch in the FPGA and using the ALE (Address Latch Enable) signal. The FPGA uses an in-system programmable Flash memory (4M$\times$16 bits). There are two serial configuration PROM's provided with the FPGA which are controlled by the ATMega128L and the configuration data can be downloaded to them through the JTAG port. A number of LED's and push-buttons have been provided at the motherboard for easy test and debugging.

C. Communication Subsystem

The communication subsystem manages the data transfer and signaling (beaconing) between the sensor nodes. It includes the network protocols and the radio transceiver. The radio transceiver is a Nordic VLSI nRF2401 that can deliver a maximum data rate of 1 Mb/s. It consists of a fully-integrated frequency synthesizer which can generate a frequency range of 2.4-2.5 GHz in the ISM band, a power amplifier, a crystal oscillator (it uses a 16 MHz external crystal), and a GFSK3 modulator. The output power and the frequency channels of the radio transceiver are fully programmable through the 3-wire (SPI) interface. The antenna used for radio transmission and reception is an omni-directional stub antenna. The transmission range of the radio transceiver has been measured to be 80 meters under ideal conditions.

D. Power Supply Subsystem

The power-supply subsystem comprises 3 or 4 AAA-sized batteries and a collection of DC-DC converters to service the entire sensor node. The voltages generated by the DC-DC converters are: 3.0V, 3.0V-Flash, 2.5V, and 1.2V. The 3.0V supplied to the radio transceiver is filtered through a low-pass passive LC filter (f_c=1.5 MHz) to generate 3.0V Analog which is also used by the ATMega128L for its analog I/O. The two serial configuration PROM's for the FPGA use two supplies: 3.0V-Flash and 2.5V. An optional 2.5V is also reserved for the radio transceiver for future use.

In the next section we describe the details of our functional test methodology which is hierarchical in nature as it combines board-level, as well as, component-level testing. The board-level test is performed by synthesizing and implementing a parameterizable Test Controller in the on-board FPGA which tests all the peripheral interfaces on the board while component-level testing combines a March-like algorithm for testing the on-board Flash memory and the software-based

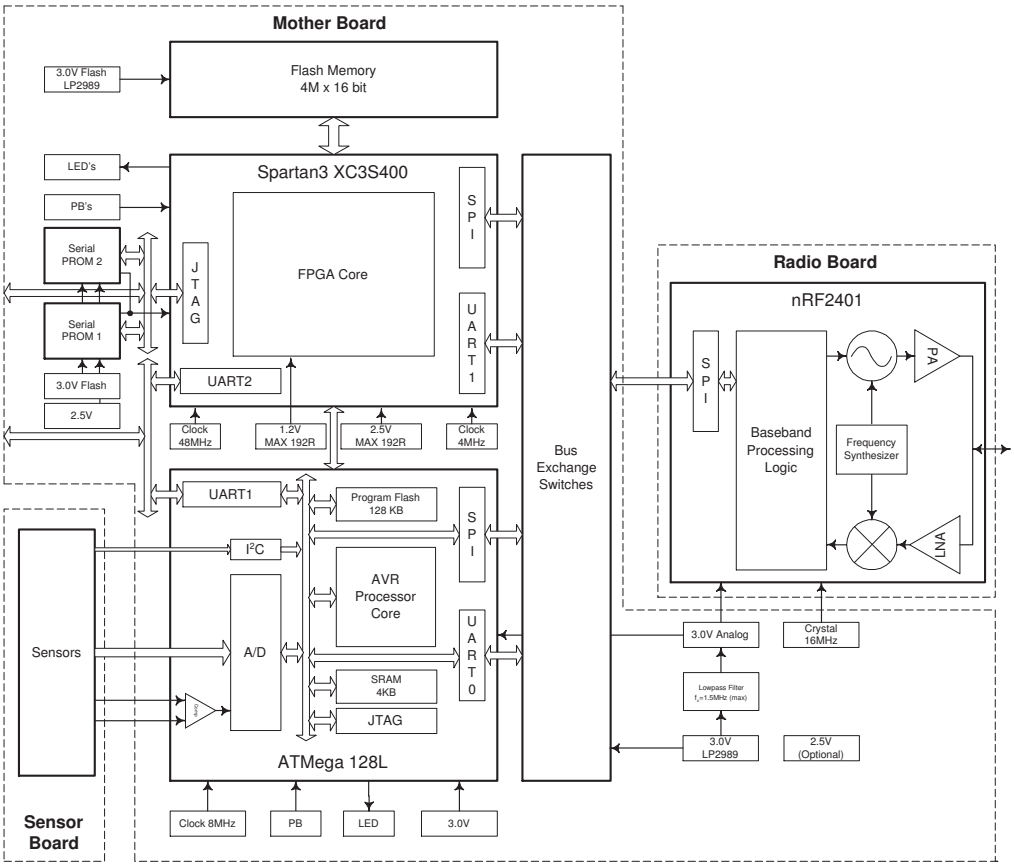


Fig. 1. *Hogthrob Sensor Node Development Platform*

self-test (SBST) technique for testing the embedded microcontroller.

III. Test of FPGA Interfaces

The use of an FPGA along with a microprocessor/microcontroller is an extremely attractive option in wireless sensor node platforms for implementing application-specific logic and/or for embedding coprocessor and DSP cores, on-chip memory blocks, peripheral devices and busses and supporting various (differential or single-ended) I/O standards because FPGA-based System-on-Chip (SoC) architectures can achieve higher integration levels in low-power, low-cost electronic systems like wireless sensor nodes. By using the existing large family of synthesizable IP (Intellectual Property) blocks combined with readily-available software drivers and libraries, the FPGA-based SoC approach allows for the rapid development of hardware and its prompt adaptation to a large variety of applications. Integration of application-specific logic blocks designed by users is facilitated by well-defined master and/or slave interfaces to the peripheral busses of embedded processors.

Likewise, the FPGA on the Hogthrob Sensor Node development platform has the following interfaces:

- FPGA-JTAG Interface
- FPGA-Push Button Interface
- FPGA-LED Interface
- FPGA-Sensor Board Interface
- FPGA-Radio Transceiver Board Interface
- FPGA-UART Interface
- FPGA-AVR Processor Interface
- FPGA-Flash Memory Interface

The interfaces listed above have to be tested for process-induced interconnect-related faults some of which are:

- short(s) between the adjacent signal traces on the impedance terminations provided or at the balls of the BGA package of the FPGA.
- cuts(s) in the signal traces due to PCB manufacturing defects (e.g., overetching) or due to board mishandling.
- improper component assembly due to which the balls of the BGA package or the pins of the integrated circuit chip are not soldered properly and do not make contact to the pads on the board.
- cross-talk between the signals due to their proximity which hampers the connectivity at high signal edge rates.
- signal integrity issues due to long trace lengths and inadequate terminations.

In order to test all the FPGA interfaces for the faults mentioned above, we have designed and synthesized a parameterizable FPGA-based Test Controller. Apart from reducing the number of LED's or Logic Analyzer channels required for

monitoring the test response of the signals comprising each interface, the advantage of such an approach for testing the interconnects in the board interfaces is that the whole process of generating the VHDL code and the user constraints file for the FPGA can be automated using the `Perl` scripts and a spreadsheet (e.g., `Excel`) file (the I/O pin configurations of the FPGA can be stored in a tabular form by importing the net data directly from the CAD tool), thus, eliminating implementation errors for sensor node designs with large number of interconnects.

The **FPGA-JTAG** Interface is tested by connecting the FPGA program download cable to the JTAG connector on the motherboard and the program download software can detect all the devices linked through the JTAG chain on the motherboard confirming the proper functioning of the FPGA-JTAG Interface. To test the **FPGA-Push Button** Interface, the FPGA-based Test Controller logic pulls up the I/O pins of the FPGA connected to the push buttons configuring the push buttons as switches for turning the LED's on or off while, for testing the **FPGA-LED** Interface, a binary counter has been implemented in the Test Controller which sends the bits of its count value to toggle the LED's causing them to blink at different rates.

Testing all the interconnections comprising the board interfaces can become a very challenging task for FPGA's having large number of I/O pins. Implementing a counter and a decoder in the FPGA and observing the decoder outputs with the LED's or the Logic Analyzer requires, in general, a k-bit counter for observing N signals where $k = log_2(N)$. However, for testing the **FPGA-Radio Transceiver Board** Interface and the **FPGA-Sensor Board** Interface, we have adopted a faster and more elegant approach by connecting the pins on the interface connectors in a daisy-chain fashion. The Test Controller implements a simple binary counter that sends the last bit of its count through the chain to toggle an LED on and off.

A. FPGA-UART Interface

The FPGA-UART Interface is tested by the Test Controller by implementing a *UART* core in the FPGA which receives and sends back a series of test characters from a PC through its UART.

B. FPGA-AVR Interface

For testing the FPGA-AVR Interface, an 8-bit *latch* is implemented in the Test Controller (Figure 2) which is controlled by the `ALE` signal from the AVR. An 8-bit$\times$256 *SRAM*, also implemented in the Test Controller, is controlled by the `WRI` and `RDI` signals from the AVR. The AVR sends the address and data on the multiplexed `DA[0:7]` bus for writing to the SRAM along with the `ALE` and `WRI` signals. The *latch* demultiplexes (separates) the address from the data and stores the data at the separated address in the *SRAM*. The data stored in the *SRAM* is read back by the AVR and checked for consistency to ensure proper operation of the FPGA-AVR Interface.

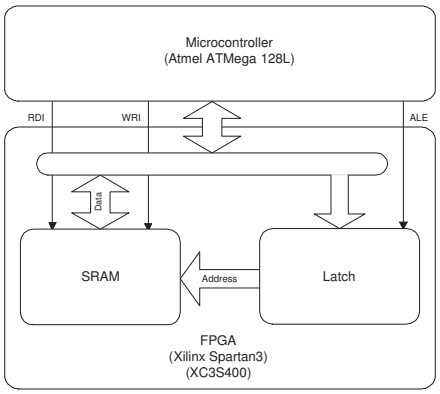


Fig. 2. *FPGA-AVR Interface*

C. FPGA-Flash Memory Interface

For testing the FPGA-Flash Interface, the Test Controller implements a *Flash Controller* (Figure 3) which performs a March-like Flash test and consists of a *Controller* block which is a programmable state machine (Figure 4) that receives the `READY` and `READ/WRITE` commands from the *Sequencer* block. The *Generator* block generates the program commands and the address and the data for programming the Flash after receiving appropriate control signals from the *Controller* block. The data written to the Flash is read back by the *Flash Controller* which is compared in the *Monitor* block which flashes an LED if the read data is the same as the data written at the corresponding address.

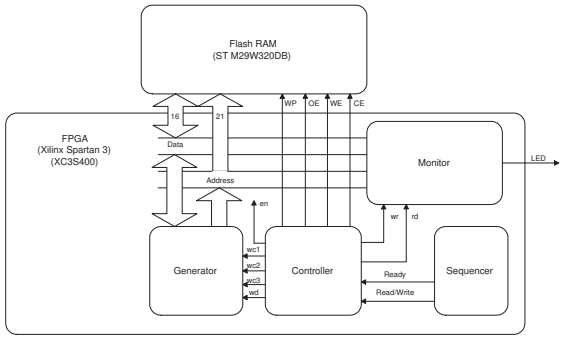


Fig. 3. *FPGA-Flash Interface*

IV. Test of Microcontroller & Interfaces

For testing the Atmel ATMega128L embedded microcontroller, we have used the Software-Based Self Test (SBST) approach as described in [5]. SBST methodology is a technology-independent, component-based processor test development strategy that uses a divide-and-conquer approach by identifying regular structures and targeting individual processor components (categorized as *functional*, *control* and *latent*) for structural (stuck-at) faults and defining different test priorities for the processor components. It combines

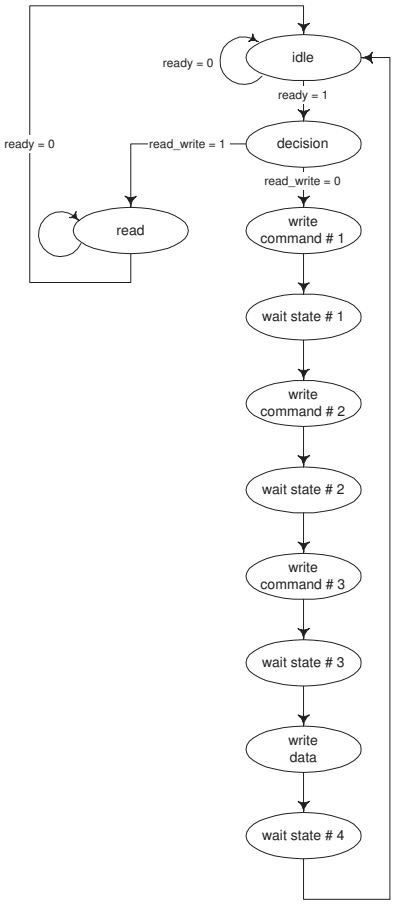


Fig. 4. *Flash Controller State Diagram*

the desirable characteristics of functional testing (using the processor instruction set for test development at a higher abstraction level) with an appropriate use of RTL information. It applies, both, to the case where low-level processor netlist is not available (COTS or 'hard' IP versions) or in the case where technology remapping is required. Another advantage of the SBST approach is that it is non-intrusive in nature since it exploits the embedded processor functionality and instruction set to carry out self testing.

According to the SBST methodology described in [5], the functional components of the processor (*computational* - arithmetic and logic modules, *interconnect* - multiplexers, and *storage* - register file and registers) have the highest test priority for test development since their size dominates the processor area and they demonstrate good controllability and observability (i.e., they provide easy and full accessibility).

We have developed a library of component test routines. Each of these test routines exercises different architectural features of the processor and generates small deterministic tests for most of the functional processor components. An important consideration while developing these test routines has been to perform collateral test coverage of non-targeted processor components (e.g., while testing the ALU, mutliplexers and the control unit are also tested).

As shown in Figure 1, the ATMega128L microcontroller

Test Routine	Description
`nop`	tight loop of the no operation instruction
`idle`	idle mode of the ATMega128L
`power-save`	power-save mode of the ATMega128L
`power-down`	power-down mode of the ATMega128L
`add`	tight loop of the add instruction stored in the register file
`add-mem`	tight loop of the add instruction stored in program memory
`hamming`	Hamming encoding and decoding

TABLE I
SBST Routines for the AVR core of the ATMega128L Microprocessor

Test Routine	Description
`blink`	blink the on-board LED's
`button`	notify an on-board push-button press
`echo`	echo a typed character
`ADC`	print the value converted by the ADC
`nRF2400`	set a Tx/Rx pair of sensor nodes and make them communicate

TABLE II
SBST Routines for the Peripheral Components of the ATMega128L Microprocessor

chip comprises an AVR core and a set of peripherals. The set of test routines developed for testing the AVR core are listed in Table I. The test programs for the AVR have been written in the TinyOS. A brief description of the test routines for testing the peripheral components is given in Table II.

The **program upload port** of the ATMega128L has been tested first by uploading and downloading the test patterns for the March-type test to the on-chip **Flash memory** through the **UART0**. Each of the tests has been carried out by a single test routine uploaded to ATMega128L. The tests have been carried out by connecting the motherboard of the Hogthrob platform to a terminal emulator program on a PC via the **UART1** and a level converter.

The `blink` and the `button` test routines test the **I/O ports** and the corresponding registers of ATMega128L as well as the interconnections between the microcontroller chip and the on-board LED and Push-Button interface while the `nRF2400` test routine tests the **SPI** interface and the corresponding register set of ATMega128L as well as the interconnection between the microcontroller chip and the radio transceiver interface. Similarly, the **ADC** test routine tests the on-chip A/D converter and the interconnection between the microcontroller chip and the analog sensors interface.

As most of the present wireless sensor node designs do not contain reconfigurable logic because of power and/or cost considerations, for testing such sensor node designs, the test controller routines can be implemented in the microcontroller. In this case, the microcontroller can undergo a self test first by the SBST method described above and the test controller routines can be executed later for testing the rest of the system.

V. Conclusions

We have developed a hierarchical, at-speed, functional test methodology and applied it successfully to test a custom-built wireless sensor node development platform. This test methodology, though unique in its approach, extends earlier

work in this area and can be applied, in general, for testing all types of wireless sensor nodes . A significant contribution of our work is a unified test methodology for wireless sensor nodes that combines, as well as, extends various component-level and board-level test techniques and exploits the on-board programmable logic for implementing a Test Controller that has a strategic access to all the board-level and component-level interfaces.

VI. Work in Progress

The prototype sensor node development platform is presently undergoing field trials and, once successful, we plan to build a sensor node prototype which will be more compact and more customized and to extend the off-line functional testing methodology described here into a self-contained, low-power, on-line self test method in order to introduce fault-tolerance with graceful performance degradation in the deployed sensor network. The ultimate goal is to shrink the sensor node to a single system-on-chip costing less than a couple of Euros.

Acknowledgements

The work presented in this paper has been funded by the Danish National Research Council (STVF) project titled ”Hogthrob – Networked On-a-Chip Nodes for Sow Monitoring” (STVF 2059-03-0027). The cooperation of Martin Leopold and Phillipe Bonnet (Department of Computing Sciences, University of Copenhagen, Denmark) and Martin Hansen (Valby Design Center, I/O Technologies A/S, Denmark) is gratefully acknowledged.

References

[1] A. Krstic, W.-C. Lai, L. Chen, and S. Dey, “Embedded Software-based Self Test for Programmable Core-based Designs,” *IEEE Design and Test of Computers*, vol. 19, no. 4, pp. 18–27, July-August 2002.

[2] K. P. Parker, “System Issues in Boundary-Scan Board Test,” in *Proceedings of the IEEE International Test Conference*, 2000, pp. 724–728.

[3] Jim Webster, “Panel Position Statement: How are we going to test SoC’s on PCB?” Proceedings of the IEEE International Test Conference, 2005.

[4] The Hogthrob Project, “http://www.hogthrob.dk.”

[5] N. Kranitis, A. Paschalis, D. Gizopoulos, and G. Xenoulis, “Software-based Self-Testing of Embedded Processors,” *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 461–475, April 2005.

[6] J.-C. Yeh, C.-F. Wu, K.-L. Cheng, and Y.-F. Chou, “Flash Memory Built-In Self Test using March-Like Algorithms,” in *Proceedings of the IEEE International Workshop on Electronic Design, Test and Applications (DELTA’02)*, 2002, p. 5 pp.

[7] R. Zhang, Z. Zilic, and K. Radecka, “Energy-Efficient Software-based Self-Test for Wireless Sensor Network Nodes,” in *Proceedings of the 24th IEEE VLSI Test Symposium (VTS’06)*, 2006, p. 6 pp.

[8] Univeristy of California, Berkeley, “TinyOS Community Forum,” http://www.tinyos.net.

Chapter 10

System-Level Modeling and Simulation of MEMS-based Sensors

Kashif Virk, Mohammad Shafique, Jan Madsen and Aric Menon. System-Level Modeling and Simulation of MEMS-based Sensors. *Proceedings of the IEEE International Multi-Topic Conference* (INMIC'05), December 2005. Pages: 1-6. Published.

System-Level Modeling and Simulation of MEMS-based Sensors

Kashif Virk,
Jan Madsen
System-on-Chip Group
Department of Informatics & Mathematical Modeling
Technical University of Denmark, Denmark
email: {virk, jan}@imm.dtu.dk

Mohammad Shafique,
Aric Menon
Micro/Nano Tribology & Modeling Group
Department of Micro & Nano Technology
Technical University of Denmark, Denmark
email: {msq, am}@mic.dtu.dk

***Abstract*—The growing complexity of MEMS devices and their increased used in embedded systems (e.g., wireless integrated sensor networks) demands a disciplined aproach for MEMS design as well as the development of techniques for system-level modeling of these devices so that a seamless integration with the existing embedded system design methodologies is possible.**

In this paper, we present a MEMS design methodology that uses VHDL-AMS based system-level model of a MEMS device as a starting point and combines the top-down and bottom-up design approaches for design, verification, and optimization. The capabilities of our proposed design methodology are illustrated through the design of a microaccelerometer.

I. Introduction

Integrated Microsystems and their subset – MicroElectroMechanical Systems (MEMS) – are inherently complex in nature. The level of their complexity can be realized from the fact that these systems involve coupled energy domains (e.g., electrical, mechanical, magnetic, fluidic, optical, etc.) and their signal conditioning units typically involve continuous-time (analog) and discrete-time (digital) electronic domains or a mixture of both (mixed-signal). The prototyping of these systems, using the available manufacturing techniques is usually very expensive. Therefore, the exising "build-and-test" approach for these systems has to be replaced by a systematic design methodology that introduces design hierarchy and information sharing across the domain dichotomies. As a part of design methodology, modeling and simulation of MEMS-based systems play an important role in reducing the number of design iterations and their time-to-market.

The design methods of MEMS have traditionally been viewed from either a bottom-up or a top-down perspective.

In the bottom-up approach, which, presently, is the most common design approach among the MEMS design communities, the idea of a MEMS device is conceived and the necessary physical-level modeling on the device design is conducted to establish its physical characteristics. However, the computational resource requirements associated with physical-level modeling render it an impractical approach for modeling the entire system. Therefore, the physical-level modeling techniques are only employed to analyze the physical characteristics of MEMS device structures and to generate the data necessary to create a reduced-order model of the device[1]. The reduced-order modeling of the MEMS device, alongwith the necessary signal conditioning and control electronics, is then conducted to detemine its proper functioning at the device level. System-level modeling is then carried out to determine the potential impact the device will have on the whole system.

On the other hand, in the top-down approach, the critical system parameters are first determined from the system-level (reduced-order) analytical equations governing the system behavior regardless of the implementation options or the process technology to be used. After determining the critical system parameters, the implementation details and the specific process technologies are considered through the use of device-level reduced-order models. Modeling at the device level involves a MEM structure with or without the signal conditioning and control electronics. At the device-level, reduced-order modeling allows the designers to determine what boundary and load conditions will be placed on individual components. After device-level modeling, more detailed physical-level modeling (3D Modeling) allows the designer to examine a structure's response to a particular physical environment in finer detail.

In this paper, we mainly focus on system-level modeling of MEMS-based sensors using VHDL-AMS as it supports multi-domain, mixed-signal modeling capabilities needed for system-level modeling of MEMS-based systems. We propose a model-driven MEMS-based system design methodology in which the design specification is captured in a system-level model using VHDL-AMS which is subsequently refined in a step-by-step manner to yield the physical design. By back-annotating the refined design parameters obtained through physical-level simulations, the same specification-based system-level model can be used for system optimization through design-space exploration by iterating back and forth the design hierarchy until a fully optimized system design is achieved. Our proposed design methodology can be viewed as a combination of top-down and bottom-up design approaches (described above) which have been modified to allow for optimization through design-space exploration.

The rest of this paper is organized as follows: Section II provides an overview of the current research in the field of system-level modeling of integrated microsystems. Section III gives a detailed explanation of our proposed model-driven design methodology. An illustrative example elaborating the capabilities of our proposed design methodology applied to the system-level modeling and design of a microaccelerom-

[1]The techniques associated with reduced-order modeling are essentially the same as macro-modeling.

eter for wireless sensor network applications is presented in Section IV. Section V, finally, provides conclusions and the future directions of our work.

II. Related Work

HDL's[2] have been used since the 1960s to model and simulate applications as diverse as (digital and analog) electronic systems and fluid concentrations in chemical processes. Modern HDLs support the description of both behavior and structure.

Depending on the available language constructs, HDLs can be divided into digital, analog, and mixed-signal HDLs. Digital HDLs, such as VHDL or Verilog, are based on event-driven modeling techniques and use a discrete model of time. They support the modeling of digital hardware at abstraction levels, ranging from system level down to the device level. Analog HDLs support the description of systems of differential and algebraic equations (DAEs) whose solution varies continuously with time. Analog HDL's like Verilog-A, MAST, VHDL-AMS, etc. support multi-domain and mixed-signal modeling capabilities and have been effectively used for MEMS modeling. HDL-A and MAST are the proprietary languages, so their development is vendor-dependent. On the other hand, Verilog-AMS and VHDL-AMS are open-source languages.

VHDL-AMS is an informal name for a combination of two IEEE standards: VHDL 1076-1993 and VHDL 1076.1-1999. It covers most of the modeling requirements for MEMS, and sufficient work (e.g., [1]–[3]) has been done to model electromechanical, MOEMS, fluidic, magnetic, and thermal systems. VHDL-AMS supports hierarchical description of continuous, mixed-domain, and discrete, conservative and non-conservative physical systems. An overview on modeling conservative systems with analog and mixed signal is discussed in [4], [5]. At the device level, VHDL-AMS has been effectively used to predict the behavior of interacting energy domains (e.g., magnetic, mechanical, electrical, etc.) using an integrated, multiple-domain, system representation.

Several techniques have been reported for the system-level modeling of MEMS devices. At the system level, MEMS devices are modelled as lumped-parameter elements (spanning multiple energy domains) along with associated electronics (analog, mixed-signal, digital). The equations governing the device behavior are the Ordinary Differential Equations (ODEs) and the Difference Equations (DEs). A hierarchically-structured design methodology for designing suspended MEMS devices which is compatible with the standard mixed-signal ASIC design flow has been described in [6], [7] using the example of a microresonator and a microaccelerometer. The design approach takes advantage of parameterized component libraries for device layout generation and modern analog and mixed-signal hardware description languages such as VHDL-AMS and Verilog-A which allow the use of non-electrical energy domains for behavioral or system-level simulation of the MEMS device. The design methodology also provides design-space exploration capability.

In [8], the general aspects concerning the design automation for microsystems (in particular, MEMS devices) are considered but no clear design methodology has been presented. In [9], a modular design methodology for suspended MEMS has been presented that uses circuit-level behavioral simulation, schematic-driven layout generation, and system-level simulation in VHDL-AMS and Verilog-AMS. A comprehensive, multi-domain, multi-language system-level modeling of Systems-on-Chip embedding MEMS devices (using SystemC and VHDL-AMS) has been reported by [10] but it does not complement the system-level model with a concrete and workable MEMS design methodology.

The top-down design of MEMS and the underlying design challenges have been discussed by [11], [12] whereas [13] discuss the bottom-up design methodology and system-level modeling using VHDL-AMS. [14] uses top-down design approach for device design and bottom-up approach for design verification. In this paper, we extend this approach to include design optimization through back annotation of refined device parameters obtained by FEM analysis at the physical level into the system-level model described in VHDL-AMS.

III. Model-Driven MEMS Design Methodology

As mentioned above, to specify, design, and implement a complex MEMS-based sensing device, it is modeled at four levels of abstraction: process-level, physical-level, device-level, and system-level. Physical-level modeling involves numerically solving the equations of physics governing the system behavior using numerical solvers such as the Finite-Element Method (FEM), Boundary Element Method (BEM), etc. Device-level modeling involves reduced-order modeling through the generation of macro-models from the physical-level models using the macro-modeling solvers. System-level modeling techniques involve block diagram-based system representation (e.g., Simulink) or it may also involve reduced-order modeling using HDL's (SystemC-AMS, VHDL-AMS, etc.) or Parametric Design Libraries[3], etc. We propose a model-driven MEMS design methodology (see Figure 1) that supports component-based design accompanied by substantial component reuse.

Starting from a design concept, a system-level model of a MEMS device is constructed for functional simulation of the design concept and for design-space exploration by examining and changing the behavioral and performance characteristics of the design concept till it meets the desired system specifications.

The key design parameters are extracted from the system-level model to construct a reduced-order model at the device level. The schematic-based, circuit-level reduced-order models involve lumped-parameter device models having few degrees of freedom with analytical or semi-analytical equations describing the behavior of the components comprising the MEMS device (e.g., beams, plates, combs, etc.). These models

[2]Hardware Description Languages (HDL's) are the programming languages specifically designed for describing the behavior of physical devices and processes. Models written in an HDL are used as input to a suitable simulator to analyze the behavior of the devices.

[3]For example, in the Coventor design environment, a 3D model of a MEMS device can also be generated from a parametric design library-based model.

are written in an HDL and compiled into a design library called the parameterized design library. A MEMS device is composed by connecting together the required components.These device models can analyze a complex device behavior in a very short time.

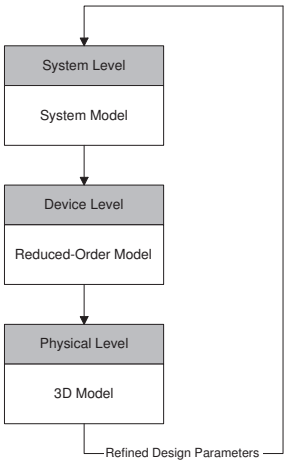


Fig. 1. System-Driven MEMS Design Methodology

After reduced-order modeling, a 3-D device structure is obtained at the physical level which is subjected to FEM-based analysis to extract the refined design parameters which are back-annotated into the system-level model for design refinement. This process is iterated till a final optimized device design is obtained.

IV. Illustrative Example: Microaccelerometer Modeling & Design

To illustrate the capabilities of our proposed MEMS design methodology, a system-level model for a capacitive microaccelerometer has been developed at the system level using VHDL-AMS which allows direct simulation of mechanical and analog and digital electric sub-systems in their respective domains without any analogy transformation.

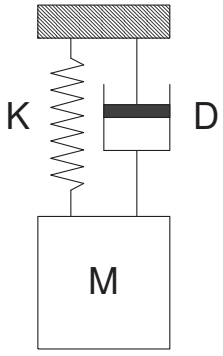


Fig. 2. Lumped-Parameter Model of Microaccelerometer

If the proof mass of the sensing element of a microaccelerometer has a mass of M, the suspension beams have an effective spring constant of K, and there is a damping factor D, affecting the dynamic movement of the mass, the microaccelerometer can be represented by a second-order mass-damper-spring system (see Figure 2). If an external force, F, displaces the support frame relative to the proof mass, the internal stress in the suspension spring changes. Both, this relative displacement and the suspension-beam stress can be used as a measure of the external force. By using Newton's second law [15]:

$$M\ddot{x} + D\dot{x} + Kx = F$$

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = a$$

In the above equation, x is the proof mass displacement, $\omega_n = \sqrt{\frac{K}{M}}$ is the natural resonance frequency, $\zeta = \frac{D}{2M\omega_n}$ is the damping factor, and $Q = \sqrt{\frac{KM}{D}}$ is the quality factor,

The resonance frequency, ω_n, of the structure can be increased by increasing the spring constant, K, and decreasing the proof mass, M, while the quality factor, Q, of the device can be increased by reducing the damping factor, D, and by increasing the proof mass, M, and spring constant, K.

The static sensitivity of the microaccelerometer is:

$$S_{static} = \frac{x_{static}}{a} = \frac{1}{\omega_n^2}$$

The static response of the device, i.e., its static sensitivity, S_{static}, can be improved by reducing its resonant frequency, ω_n.

The primary mechanical noise source for the device is due to the Brownian motion of the gas molecules surrounding the proof mass and the Brownian motion of the proof mass suspension or anchors. The total noise equivalent acceleration (TNEA) is:

$$TNEA = \frac{\sqrt{4k_bTD}}{M} = \sqrt{\frac{4k_bT\omega_n}{QM}}$$

where,

k_b is the Boltzmann's constant and T is the absolute temperature.

Thus, to reduce the mechanical noise, the quality factor, Q, and the proof mass, M, have to be increased.

For the system response to be linear, a, has to be less than $0.3\omega_n$. This sets the maximum bandwidth, B_{max}, of the microaccelerometer.

$$2\pi B_{max} \leq 0.3\omega_n \Rightarrow \omega_n \geq 20.944 B_{max}$$

For the least amplitude distortion and for the output to follow the input over the widest input frequency range, the system has to be critically damped. This implies that:

$$\zeta = 0.707$$

or

$$\frac{D}{2M\omega_n} = 0.707$$

The minimum detectable acceleration can be obtained from the expression for TNEA:

$$a_{min} = \frac{\sqrt{4k_bTD}}{M} = \sqrt{\frac{4k_bT\omega_n}{QM}}$$

The maximum detectable acceleration can be obtained from $S_{dynamic}$ as:

$$a_{max} = \frac{Kg_{max}}{M}$$

However, g_{max} cannot be below $\frac{g}{3}$, because after that pull-in voltage ($V_p = \frac{2}{3}\sqrt{\frac{2Kg}{A\epsilon}}$) kicks in and electrodes collide with each other.

The difference between a_{max} and a_{min} gives the dynamic range of the microaccelerometer. There is a big tradeoff between dynamic range and sensitivity.

In the most general case, the proof-mass motion can have six degrees of freedom. But, typically, in a unidirectional accelerometer, the geometrical design of the suspension is such that one of these is dominant and the device has low off-axis sensitivity. The cantilever support has been one of the early popular suspension support designs, due to its simplicity, lower spring constant, and internal stress relief of the beams. However, this configuration results in a larger off-axis sensitivity unless the device is fully symmetric. Symmetric, full-bridge supports result in a very low off-axis sensitivity. By using a crab-leg or folded-beam configuration in a full-bridge support, the residual stress of the beams can also be relieved keeping the spring constant unchanged due to tensile and compressive stresses. The spring constant for the folded beam configuration employing straight truss is:

$$K = (\frac{\pi^4}{6})[\frac{Ewh^3}{(2L_1)^3+(2L_2)^3}]$$

If parallel-plate estimates are used to get the correct order of magnitude of capacitances:

$$C_{sense} = N\frac{\epsilon_0 hl}{g_0 \pm x}$$

The plates of a parallel-plate capacitor attract each other with an electrostatic force of:

$$F = \frac{\epsilon_0 hl}{2g_0^2}$$

The mass of the proof mass with attached cantilever electrodes can be estimated from the device dimensions.

The damping factor is a difficult quantity to calculate because the effect of *squeezed-film* damping between the fingers must be added to the *Couette flow* beneath the proof mass as it displaces. Further, if the aspect ratio of the air gaps between the fingers is low, even squeezed-film damping estimates are inaccurate. The damping factor, D_1, obtained from the Couette flow[4] is:

$$D_1 = \eta\frac{A}{h}$$

where,

η is the viscosity of the surrounding air/gas. The damping factor, D_2 obtained from the squeezed-film model is:

$$D_2 = N\eta\frac{lw^3}{\pi^4 h_0^3}$$

The total damping, D, is the sum of D_1 and D_2.

The differential capacitors have high sensitivities and can be configured to give a linear response and are, therefore, preferred for many applications. The differential capacitors have the virtue of cancelling many effects to first order, providing a signal that is zero at the balance point and carries a sign that indicates the direction of motion. From the system point of view, a differential capacitor accomplishes linearization about the balance point. Consider an interdigitated parallel-plate differential capacitor with the gap of the upper capacitor g_1 and that of the lower capacitor g_2. Assuming an equal area of both capacitors, a voltage $+V_s$ is applied to the upper plate and a voltage, $-V_s$ is simultaneously applied to the lower plate. The voltage appearing at the voltage divider output is:

$$V_0 = -V_s + \frac{C_1}{C_1+C_2}2V_s = \frac{C_1-C_2}{C_1+C_2}V_s$$

since the areas are equal:

$$V_0 = \frac{g_1-g_2}{g_1+g_2}V_s$$

If the two gaps are equal, the output voltage is zero. However, if the middle plate moves so that one gap is larger than the other, the output voltage is a linear function of this change. The resulting sensing element output is a square wave with amplitude propotional to the displacement and, hence, the acceleration magnitude. The phase of the output square wave relative to the excitation determines the acceleration polarity which measures the unbalance in the differential capacitor. The output is amplified, synchronously demodulated, low-pass filtered and digitized with a $\Sigma - \Delta$ A/D converter to give the output value. Since the demodulator is phase synchronized with the excitation signal, the output signal polarity correctly indicates the direction of the applied acceleration [16].

```
library ieee;
use ieee.math_real.all;
use ieee.mechanical_systems.all;
use IEEE.electrical_systems.all;
use work.all;

entity testbench is
end entity testbench;

architecture ideal of testbench is
terminal telect1, telect2 : electrical;
terminal ttrans1, ttrans2 : translational;
quantity tq : real;
terminal tinput1, tinput2 : electrical;

begin
    ins_spring: entity work.spring(linear)
    port map (trans1 => ttrans1,trans2 => ttrans2);

    ins_mass: entity work.mass(ideal)
    port map (trans1 => ttrans1, trans2 => ttrans2, elect1 => tinput1,
              elect2 => tinput2, q_v => tq);

    ins_damper: entity work.damper(ideal)
    generic map (D => 0.0)
    port map (trans1 => ttrans1,trans2 => ttrans2);

    ins_parallel_plate: entity work.parallel_plate(ideal)
    port map (elect1 => telect1, elect2 => telect2, trans1 => ttrans1,
              trans2 => ttrans2, q_c => tq);

    ins_v_pulse: entity work.v_pulse(ideal)
    generic map (pulse => 5.0, width => 50ms, period => 100ms)
    port map (pos => telect1, neg => electrical_ref);

    ins_forcepulse: entity work.forcepulse(ideal)
    generic map (pulse => 1000.0, width => 500ms, period => 1000ms)
    port map(trans_pos => ttrans1, trans_neg => translational_ref);
end architecture ideal;
```

Fig. 3. Top-level VHDL-AMS Code for System-level Model

A surface micromachined, single-axis, lateral capacitive microaccelerometer sensing element consists of a movable beam (seismic mass), suspended by two spring tethers on either end. Movable fingers are attached to the mass. The fingers establish, together with fixed plates, capacitances that are evaluated by an electronic circuit. If the seismic mass is moved by an external force, the capacitances depend on this force. The structure is highly regular. Similar microelectro-

[4]Steady viscous flow between parallel plates, one of which is moving parallel to the other, is called Couette flow

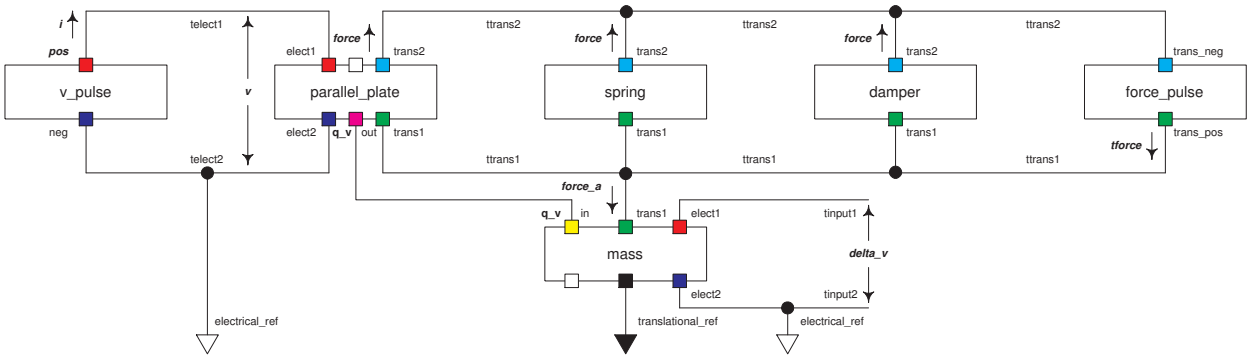


Fig. 4. System-Level VHDL-AMS Model Block Diagram

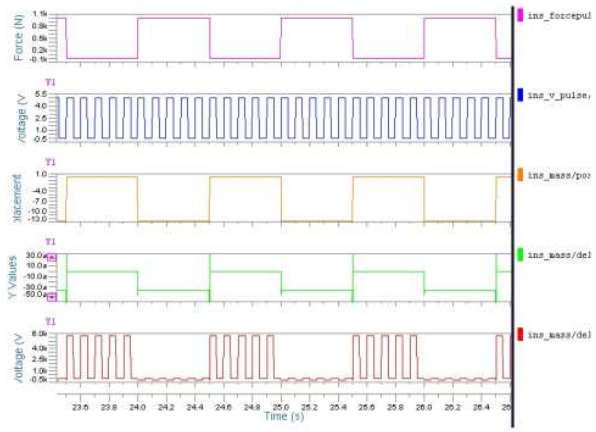


Fig. 5. System-Level Transient Simulation in VHDL-AMS

mechanical devices are used in force-balanced accelerometers like the ADXL series from Analog Devices and Siemens.

Using the proposed model-driven design methodology, general system-level modeling of the microaccelerometer has been performed in VHDL-AMS using the above equations, as well as the mechanical relations describing the spring constant and the damping factor as a function of the device geometry and the ambient pressure. The basic functionality of the microaccelerometer has been simulated using VHDL-AMS (see Figure 3) A block diagram description of the VHDL-AMS model is given in Figure 4. The time-domain characteristics are simulated for different modeling approaches using simulation and are shown in Figure 5.

Further, the first-order device design optimization has been performed using the same equations, while the final microaccelerometer sensing element design has been simulated and optimized using the commercially-available finite-element method (FEM) solvers in the Coventer software package.

Figure 6 shows the schematic drawing of the sensing element and the electronic circuitry of such a microaccelerometer built using components from the parametrized design library of Saber and MAST in the Coventer design environment. The device-level model consists of mechanical beams of different dimensions. The electrostatic forces are modeled by comb models (see Figure 7). The design parameters are listed in Table I. The advantage of this (schematic-based) approach is an easy combination of these MEMS primitives with other user-defined models. Therefore, for functional simulation, behavioral models of the electronic subsystems were used. Figure 8 shows the displacement of the sensing element at the first resonance frequency.

TABLE I
DESIGN PARAMETERS

Proof Mass	
Length	410μ
Width	90μ
Height	10μ
Damping Holes	
Length	10μ
Width	10μ
Depth	10μ
Number	80
Finger	
Length	160μ
Width	$10\ \mu$
Thickness	5μ
Number	22
Suspension Support	
Length1	170μ
Length2	185μ
Length3	30μ

V. CONCLUSIONS

We have proposed a model-driven MEMS design methodology which is more than a combination of the existing top-down and bottom-up design approaches as it enables MEMS design, validation, and optimization in a consistent, step-by-step manner and is compatible with the existing embedded system design methodologies. We have illustrated the capabilities of our proposed MEMS design methodology by applying it to design, simulate, and optimize a microaccelerometer sensor. System-level modeling of MEMS-based sensors is an ongoing research area with the aim to model, as accurately as

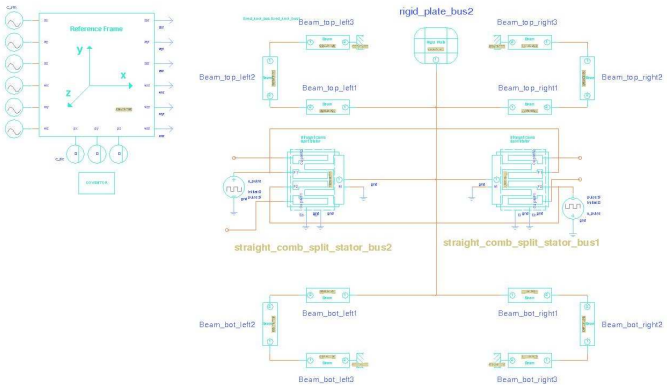


Fig. 6. Sensing Element Schematic

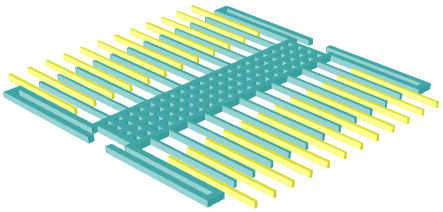


Fig. 7. Sensing Element

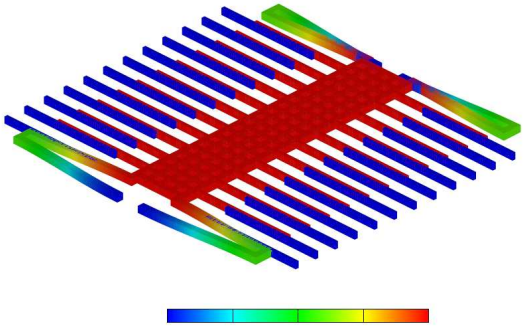


Fig. 8. Sensing Element FEM Simulation in Coventor

possible, the microsystem device behavior at the system level as well as at the lower levels of abstraction. The capabilities of the existing design tools for designing microsystems are still limited to some extent, either in the diversity of components in the parametric components library, accuracy of simulation, or in terms of simulation speed and ease of use. Lack of standards and interoperability are additional limitations. Information about macromodeling or lumped parameter modeling of microsystems is ample but patchy. Moreover, automated synthesis of microsystems from system-level models seems to be a far-off dream. The work described here gives a good insight into the system-level modeling of microsystems which can stimulate ideas about hybrid systems modeling and verification especially in the context of the emerging new area of wireless integrated sensor networks.

ACKNOWLEDGEMENTS

This work is a joint first author effort. The cooperation and support provided by the departments of Informatics and Mathematical Modeling (IMM) and Micro and Nano Technology (MIC) is gratefully acknowledged.

REFERENCES

[1] F. Pecheux, C. Lallement, and A. Vachoux, "Vhdl-ams and verilog-ams as alternative hardware description languages for efficient modeling of multidiscipline systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 2, pp. 204–225, 2005.

[2] P. Wilson, J. Ross, A. Brown, and A. Rushton, "Multiple domain behavioral modeling using vhdl-ams," *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, pp. V–644, 2004.

[3] S. Guessab and J. Oudinot, "Modeling of a piezoelectric device with shocks management using vhdl-ams," *Behavioral Modeling and Simulation Conference, 2004. BMAS 2004. Proceedings of the 2004 IEEE International*, pp. 19–24, 2004.

[4] Christen, Ernst and Bakalar, Kenneth, "VHDL 1076.1 - analog and mixed-signal extensions to VHDL," *European Design Automation Conference - Proceedings*, pp. 556–561, 1996.

[5] Ernst Christen and Kenneth Bakalar, "VHDL-AMS - A Hardware Description Language for Analog and Mixed-Signal Applications," *IEEE Transactions on Circuits and Systems*, vol. 46, no. 10, pp. 1263–1272, October 1999.

[6] T. Mukherjee and G. K. Fedder, "Structured design of microelectromechanical systems," *Proceedings of Design Automation Conference (DAC)*, pp. 680–685, 1997.

[7] T. Mukherjee, G. K. Fedder, and R. Blanton, "Hierarchical design and test of integrated microsystems," *IEEE Design and Test of Computers*, vol. 16, no. 4, pp. 18–27, 1999.

[8] J. van Kuijk, G. Schropfer, and M. Dasilva, "Design automation for mems/mst," *Coventor BV. The Netherlands*, 2004.

[9] T. Mukherjee, "Mems design and verification," *IEEE International Test Conference (TC)*, pp. 681–690, 2003.

[10] Z. Juneidi, K. Torki, S. Martinez, G. Nicolescu, B. Courtois, and A. Jerraya, "Global modeling and simulation of system-on-chip embedding mems devices," *ASIC, 2001. Proceedings. 4th International Conference on*, pp. 666–669, 2001.

[11] G. K. Fedder, "Top-down design of mems," *2000 International Conference on Modeling and Simulation of Microsystems - MSM 2000 and 2000 International Conference on Modeling and Simulation of Microsystems - MSM 2000*, pp. 7–10, 2000.

[12] M. S. McCorquodale, F. H. Gebara, K. L. Kraver, E. D. Marsman, R. M. Senger, and R. B. Brown, "A top-down microsystems design methodology and associated challenges ." pp. 20 292–20 296, 2003.

[13] P. Schwarz and P. Schneider, "Model library and tool support for mems simulation," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 4407, pp. 10–23, 2001.

[14] Joachim Haase and Jens Bastian and Sven Reitz, "VHDL-AMS in MEMS Design Flow," in *Proceedings of Forum on Specification and Design Languages*, September 2002.

[15] Senol Mutlu, University of Michigan, Ann Arbor, "Surface Micromachined Capacitive Accelerometer with Closed-Loop Feedback," http://www.eecs.umich.edu/ smutlu/projects.html, pp. 1–11.

[16] Howard Samuels, "Single- and Dual-Axis Micromachined Accelerometers," *Analog Dialogue*, vol. 30, no. 4, pp. 3–5, October-December 1996.

Chapter 11

HW/SW Codesign of Kalman Filter for a Wireless Sensor Network Application

---

Kashif Virk and Jan Madsen. HW/SW Codesign of Kalman Filter for a Wireless Sensor Network Application. Not Published.

HW/SW Codesign of Kalman Filter for a Wireless Sensor Network Application

Kashif Virk and Jan Madsen
System-on-Chip Group
Computer Science & Engineering Section
Department of Informatics & Mathematical Modeling
Technical University of Denmark, Lyngby 2800, Denmark
email: {virk, jan}@imm.dtu.dk

***Abstract*—This paper describes the details of a cycle-accurate model of a coprocessor for Kalman filtering which is a standard DSP tool for combining information from many sensors as well as low-pass filtering, amplification, etc. A properly-designed Kalman Filter allows observation of only a few quantities, or measured outputs and reconstruction or estimation of the full internal state of a system.**

We consider a wireless sensor network application in which accelerometer data from field experiments on sows are analyzed for acceleration patterns and an automatic classification method based on a Multi-Process Kalman Filter has been devised. However, the practical implementation of such analysis method poses problems because Kalman Filter implementation for real-time applications is computation-intensive in software and resource-demanding in hardware due to matrix multiplication and inversion operations. Therefore, we have developed a design flow for design-space exploration using HW/SW Codesign to select the optimum implementation and implemented an FPGA-based cycle-accurate model of a coprocessor block for Kalman filtering.

I. Introduction

Filters are commonly used to extract a desired signal from a backgroud of random noise or deterministic interference. In the statistical approach to the solution of the linear filtering problem, knowledge of certain statistical parameters of the useful signal and unwanted additive noise (e.g., mean and correlation functions) is assumed. The problem is to design a linear filter with the noisy data as input and the requirement of minimizing the effect of the noise at the filter output according to some statistical criterion.

Consider the following situation: An original signal s(t) is transmitted through an information channel (cable, wireless channel, storage medium). The received signal x(t) is impaired by two different effects. Firstly, the channel may not have a perfect impulse (delta-function) response so that the original signal s(t) is convolved with some known impulse response g(t) to give a smeared signal v(t) = g(t)*s(t). Secondly, noise n(t) may be added to v(t) to, finally, give the signal x(t) = v(t)+ n(t) at the receiver. Our task is to find the optimal filter h(t) which, when applied to the signal x(t) produces a signal y(t) that is as close as possible to the uncorrupted signal s(t). In other words, we want to estimate the true sigal s(t).

A useful approach to this filter-optimization problem is to minimize the mean-square value of the error signal that is defined as the difference between some desired response and the actual filter output. For stationary signal inputs, the resulting solution is commonly known as the **Weiner filter**.

Wiener filters are a class of optimum linear filters which involve linear estimation of a desired signal sequence from another related sequence. They are designed to minimise the mean-square error between their output and a desired or required output.

Most of the filter design techniques are firmly based on fequency domain concepts. By contrast, Wiener filters are developed using time-domain concepts. However, the Weiner filter is inadequate for dealing with situations in which nonstationarity of the signal and/or noise is intrinsic to the problem. In such situations, the optimum filter has to be assumed to be of a time-varying form. A highly successful solution to this more difficult problem is found in the **Kalman filter**.

Sensor fusion is important in a network of sensors of different modalities. A distributed vehicle/personnel surveillance network might include seismic, acoustic, infrared motion, temperature, and magnetic sensors. The standard DSP tool for combining the information from many sensors is the Kalman Filter. The Kalman Filter is used for communications, navigation, feedback control, and elsewhere and provides the accuracy that allowed man to navigate in space and, eventually, to reach the moon and, more recently, to send probes to the limits of the Solar System.

A properly designed Kalman Filter allows one to observe only a few quantities, or measured outputs, and then reconstruct or estimate the full internal state of a system. It also provides low-pass filtering functions and amplification, and can be constructed to provide temperature compensation, common mode rejection, zero offset correction, etc.

The discrete-time Kalman Filter, useful for DSP, is a dynamical filter given by the following equation [1]:

$\hat{x}_{k+1} = \mathbf{A}(\mathbf{I} - K\mathbf{H})\hat{x} + \mathbf{B}u_k + \mathbf{A}Kz_k$

where the sensed outputs are in vector z_k, the control inputs to the system being observed are in vector u_k, and the estimates of the internal states are given by the vector $\hat{x}_k$. Note that the number of sensed outputs can be significantly less than the number of states one can estimate. In this filter, matrices $\mathbf{A}$ and $\mathbf{B}$ represent the known dynamics of the sensed system, and the sensed outputs are given as a linear combination of the states by $z_k = \mathbf{H}x_k$, where $\mathbf{H}$ is a known measurement matrix. The Kalman gain K is determined by solving a design equation known as the Riccati Equation. The Kalman Filter is

the optimal linear estimator given the known system properties and prescribed corrupting noise statistics.

A. Kalman Filter Design Equations [2]

Predict

- *Project the State Ahead:*
  $\hat{x}_k^- = \mathbf{A}\hat{x}_{k-1} + \mathbf{B}u_k$
  `p_est = A*p + B*u`
- *Project the Error Covariance Ahead:*
  $P_k^- = \mathbf{A}P_{k-1}\mathbf{A}^T + \mathbf{Q}$
  `P_cap_est = A*P_cap*A' + Q`

Correct

- *Compute the Kalman Gain:*
  $K_k = P_k^-\mathbf{H}^T(\mathbf{H}P_k^-\mathbf{H}^T + R)^{-1}$
  `K = (H'*P_cap_est)*inv(H*P_cap_est*H' + R)`
- *Update Estimate with Measurement z_k:*
  $\hat{x}_k = \hat{x}_k^- + K_k(z_k - \mathbf{H}\hat{x}_k^-)$
  `p = p_est + K*(A' - p_est)`
- *Update the Error Covariance:*
  $P_k = (\mathbf{I} - K_k\mathbf{H}P_k^-)$
  `P_cap = (I - KH)*P_cap_est`

II. HW/SW Codesign of Kalman Filter

As described above, Kalman Filter is the standard DSP tool for combining the information from many sensors as well as low-pass filtering, amplification, etc. A properly designed Kalman Filter allows one to observe only a few quantities, or measured outputs, and then reconstruct or estimate the full internal state of a system.

In the Hogthrob project, accelerometer data from the field experiments on sows were analyzed for acceleration patterns and an automatic classification method based on a Multi-Process Kalman Filter was implemented by the KVL research group [3].

However, the practical implementation of such analysis method poses problems because Kalman Filter implementation for real-time applications is computation intensive in software and resource demanding in hardware due to matrix multiplication and inversion operations.

Therefore, we developed a design flow for design-space exploration using HW/SW Codesign to select the optimum implementation and implemented an FPGA-based coprocessor block for the Kalman Filter.

A. Software Implementation

A software-based Kalman filtering algorithm was implemented in C language (code listing given below) and cross compiled to AVR processor using the GNU C cross assembler and linker for AVR processors. However, because of matrix multiplication and inversion operations, its execution on the resource-constrained AVR processor was extremely slow and, therefore, far from meeting the real-time throughput requirements of the system.

B. Hardware Implementation

Although data from the acceleration sensors was analyzed using Kalman Filters in the **R** modeling language, we selected the MATLAB language for algorithmic modeling because it offers a rich environment for DSP algorithm development and debugging and is uniquely adept with vector- and array-based waveform data at the core of DSP algorithms.

1) Design Flow: Traditional FPGA deployment of DSP algorithms can involve many steps that take an algorithm from C instructions to an FPGA-specific bit stream. To simplify rapid prototyping of DSP-in-FPGA designs, a high-level MATLAB-based algorithm synthesis package, the Xilinx AccelDSP Algorithmic Synthesis Tool, lets DSP algorithm developers create DSP blocks for Xilinx FPGA's.

AccelDSP automates floating-point to fixed-point conversion, generates synthesizable VHDL, and creates a test bench for verification. In short, it translates the MATLAB algorithms specified in the m-files into synthesizable VHDL code.

The Xilinx System Generator is a rapid prototyping tool for creating hardware DSP designs using graphical methods. With a visual programming environment that leverages the MathWorks Simulink tool and Xilinx Block Set library of predefined digital signal-processing and communications functions such as filters, fast-Fourier transforms, encoders, decoders and so on., Xilinx System Generator meets the needs of both system architects (to integrate design components) and hardware designers (to optimize implementations).

The Xilinx AccelDSP Synthesis tool augments the Xilinx System Generator by providing a seamless integration path for DSP algorithm developers, enabling the rapid creation of DSP IP blocks, directly from m-files, that enhance the Xilinx Block Set in the Xilinx System Generator. In addition, AccelDSP has optional AccelWare toolkits that complement System Generator with additional DSP IP cores optimized for Xilinx FPGA cores. The building blocks of DSP functions can drop into System Generator. AccelWare toolkits include mathematical building blocks, signal processing, communications, and advanced mathematics to implement linear algebra functions.

AccelDSP was used to explore different micro and macro architectures. A macro architecture could encompass something as simple as a divide operation where one could use a CORDIC, a Newton-Raphson, a Goldschmidt or another division technique. After this decision, the possibilities of using a pipeline or a resource-shared micro architecture were explored. That way we could trade off implementation requirements and capabilities. By automating the design flow we could develop algorithms within MATLAB, use Simulink as a software test bench and then validate our design within a real FPGA.

By using the design flow described above, We could concentrate on getting the best performance out of their algorithms and not on how to implement them on an FPGA chip.

2) Design Capture: The Kalman filter algorithm was captured with a MATLAB m-file to perform stimulus creation, algorithm evaluation, and post-processing. A listing of the m-file is given in Figure 3.

The algorithm defines matrices `R` and `I` that describe the statistics of the measured signal and the predicted behavior.

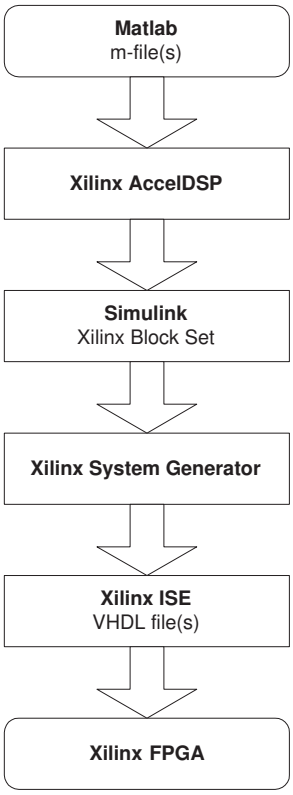


Fig. 1. *Kalman Filter Design Flow.*

```
function [S] = simple_kalman(A)
DIM = size(A,2);
persistent p P_cap
if isempty(P_cap)
P_cap = [8 0 0; 0 8 0; 0 0 8];
p = ones(DIM,1)/2;
end;
I = eye(DIM);
R = [128 0 0;0 128 0; 0 0 128];
% estimate step:
%p_est = p;
P_cap_est = P_cap+I;
% correction step:
K = P_cap_est * inv(P_cap_est+R);
p = p + K * ( A' - p );
P_cap = (I - K)*P_cap_est;
S = p';
```

Fig. 2. *The MATLAB m-file describing code for Kalman Filter.*

The last nine lines of the algorithm are the code that predicts and corrects the estimate (ref. Section **??**).

This algorithm illustrates the flexibility and conciseness of the MATLAB language. Common operators such as addition and subtraction operate on variables like the two-dimensional arrays `A` or `P_cap` without having to write loops, as one would in languages like C. Multiplication of two-dimensional arrays is automatically performed as matrix multiplication without any special annotation. MATLAB operators such as matrix transposition allow the MATLAB code to be compact and easily readable. And complex operations like matrix inversion are completed using MATLAB's extensive linear algebra capabilities.

Although such an algorithm could have been constructed as a block diagram, doing so would have obscured the algorithm structure so readily apparent in MATLAB.

3) Design-Space Exploration: With Xilinx AccelDSP [4], a first step in synthesizing a complete algorithm is to generate any major cores that are referenced - in this case, the matrix inverse indicated by the function call `inv(P_cap_est+R)`. But a matrix inverse can be implemented in many ways; the choice of which method to use depends on the size, structure, and values of the matrix. Using the matrix inverse IP core from the Xilinx AccelWare toolkit, we could choose from micro-architectures designed for different applications. These micro-architectures can be optimized for speed, area, power, or noise. In this case, the most suitable approach was to use the AccelWare QR matrix inverse core.

4) Design Optimization: The m-files were loaded into the Xilinx AccelDSP tool and they served as the "golden source" for a design flow that ultimately produced optimized implementations in Xilinx FPGAs. With the MATLAB m-file loaded into Xilinx AccelDSP, the next step was to simulate the floating-point design to establish a baseline. AccelDSP was used to convert the design to fixed-point format [5], verifying it in MATLAB. AccelDSP offered us an array of tools to help trim bits from the design and verify the fixed-point design effects like saturation and rounding. AccelDSP aided us in this process by propagating bit growth throughout the design and letting the use of directives to set constraints on bit width. This algorithmic design space exploration allowed us the attainment of the ideal quantization that minimized bit widths while managing overflows or underflows, allowing early trade-offs of silicon area versus performance metrics.

Once suitable quantization had been attained, the next step was to generate RTL for the target Xilinx device. At this point, the AccelDSP GUI was used to set constraints on the design using the following design directives to achieve further optimizations:

- Rolling/unrolling of FOR loops
- Expansion of vector and matrix additions and multiplications
- RAM/ROM memory mapping of vectors and two-dimensional arrays
- Pipeline insertion
- Shift-register mapping

Using these directives constituted hardware-based design exploration, allowing further improvement to the quality of results. In synthesizing the RTL, AccelDSP evaluated the entire design and scheduled the entire algorithm, performing necessary boundary optimization in the process.

Throughout this flow, Xilinx AccelDSP maintained a uniform verification environment through a self-checking test bench; the input/output vectors that were generated when verifying the fixed-point MATLAB design were used to verify the generated RTL. The RTL verification step also gave Xilinx AccelDSP the information necessary to compute the throughput and latency of the Kalman filter. This was essential information to assess whether the design met specifications

and was critical for achieving cycle-accurate simulation.

5) Design Implementation: Although RTL verification is a key step in the design flow, we wanted to see algorithms running in hardware. Xilinx System Generator's hardware-in-the-loop co-simulation interfaces made this a push-button flow, bringing the full power of MATLAB and Simulink analysis functions to hardware verification.

Having run RTL verification in AccelDSP, the AccelDSP design was now ready to be exported to the Xilinx System Generator by going to the "Export" pull-down menu in the AccelDSP GUI and selecting "System Generator". AccelDSP then generated a cycle-accurate System Generator block that supports both simulation and RTL code generation.

At this point, the design flow transitioned to System Generator [6], where a new block for the Kalman filter was available in the Simulink library browser. The Kalman filter block only needed to be selected and dragged into the destination model to incorporate the AccelChip-generated Kalman filter into a System Generator design. Once the AccelDSP-generated block was included in the System Generator design, a complete, system-level simulation of cycle-accurate, bit-true models could be performed to verify that the system met specifications.

The AccelDSP-generated blocks could be used for System Generator in conjunction with the Xilinx block set. Once this system-level verification step was completed, the next step in the System Generator flow was to move on to design implementation. The "Generate" step in System Generator compiled the design into hardware.

6) Design Verification: All design files generated by AccelDSP, including exported System Generator files, were verified back to the original "golden" source MATLAB m-file. AccelDSP's verification approach is based on the generation of a test bench from the MATLAB source - this test bench was applied at the RTL level within AccelDSP and could be applied in System Generator to verify the correctness of the design. Once verified in the System Generator environment, the AccelDSP-generated block could be verified using System Generator's supported methods - including HDL co-simulation and hardware-in-the-loop - to accelerate hardware-level simulation 10 to 100 times [7].

III. Conclusions

The use of HW/SW Codesign aprroach for Kalman filter implementation enabled us to select the best implementation method that could meet the system specifications which, in this case, was a hardware implementation. By using the algorithm design and hardware design tools together, we could employ the most productive means of modeling hardware for implementation and completing high complexity designs more rapidly.

Acknowledgements

The work presented in this paper has been funded by the Danish National Research Council (STVF) project titled "Hogthrob – Networked On-a-Chip Nodes for Sow Monitoring" (STVF 2059-03-0027).

References

[1] F. L. Lewis, "Wireless Sensor Networks," *Smart Environments: Technologies, Protocols and Applications, John Wiley & Sons*, pp. 1–18, 2004.
[2] Greg Welch and Gary Bishop, "An Introduction to the Kalman Filter - SIGGRAPH 2001 Course Notes," ACM, Inc., August 2001.
[3] Cecile Cornou, "Automated Monitoring Methods For Group Housed Sows, Ph.D. Thesis," KVL, KU, 2007.
[4] Eric Cigan and Narinder Lall, "Integrating MATLAB Algorithms into FPGA Designs," *XCell Journal*, Second Quarter 2005.
[5] Thomas Hill, "AccelDSP Synthesis Tool - Floating-Point to Fixed-Point Conversion of MATLAB Algorithms Targeting FPGAs," White Paper WP239(v1.0) - Xilinx, Inc., April 2006.
[6] ——, "Using MATLAB to Create IP for System Generator for DSP," White Paper WP241(v1.0) - Xilinx, Inc., April 2006.
[7] Users' Guide v9.1.0.1, "Xilinx System Generator for DSP," Xilinx, Inc., March 2007.
[8] Christian Olesen Obel, "M.Sc. Thesis," IMM, DTU, 2007.

Appendix

The C code for Kalman Filter [8] is listed below in the following pages.

```
// Implementation of the Kalman filter for one time-series of the activity
// classification algorithm for the Hogthrob Project. The purpose is to find
// the memory requirements and computational complexity of this piece of code
// and, later, optimize the memory usage.

#include <stdio.h>
#include <stdlib.h>

float at[3];    // array containing "Prior means"
float Rt[3*3];  // array containing "Prior Variances"
float ft[1];    // array containing "One-step Forecast means"
float Qt[1];    // array containing "One-step Forecast variances"
float At[3];    // array containing "Adaptive Coef. matrix"
float et[1];    // array containing "One-step Forecast error"
float mt[3];    // array containing "Filtered means"
float Ct[3*3];  // array containing "Filtered variances"

int additions = 0;
int multiplications = 0;
int divisions = 0;

// Parameter Ft contains tabularized sines and cosines. R code:
// st <- sin(((2*pi)/T)*(1:n));
// ct <- cos(((2*pi)/T)*(1:n));
// Ft <- cbind(rep(1,n),st,ct);

float Ft[3*21] = {1, 2.947552e-01, 0.9555728, \
                  1, 5.633201e-01, 0.8262388, \
                  1, 7.818315e-01, 0.6234898, \
                  1, 9.308737e-01, 0.3653410, \
                  1, 9.972038e-01, 0.0747301, \
                  1, 9.749279e-01,-0.2225209, \
                  1, 8.660254e-01,-0.5000000, \
                  1, 6.801727e-01,-0.7330519, \
                  1, 4.338837e-01,-0.9009689, \
                  1, 1.490423e-01,-0.9888308, \
                  1,-1.490423e-01,-0.9888308, \
                  1,-4.338837e-01,-0.9009689, \
                  1,-6.801727e-01,-0.7330519, \
                  1,-8.660254e-01,-0.5000000, \
                  1,-9.749279e-01,-0.2225209, \
                  1,-9.972038e-01, 0.0747301, \
                  1,-9.308737e-01, 0.3653410, \
                  1,-7.818315e-01, 0.6234898, \
                  1,-5.633201e-01, 0.8262388, \
                  1,-2.947552e-01, 0.9555728, \
                  1,-2.449213e-16, 1.0000000};

// Parameter Yt contains the time series in use. This Yt is the first
// 120 samples from EAM1 (J=k=1). This is, probably, data from the X-axis.

float Yt[120] = {0.061035156250, 0.130371093750, 0.161376953125, \
                 0.258984375000, 0.207275390625, 0.272216796875, \
                 0.354980468750, 0.357421875000, 0.358886718750, \
                 0.541015625000, 0.496093750000, 0.435302734375, \
                 0.416992187500, 0.521728515625, 0.502929687500, \
                 0.266601562500, 0.410644531250, 0.205322265625, \
                 0.238525390625, 0.384521484375, 0.419433593750, \
                 0.110595703125, 0.116943359375, 0.891601562500, \
                 0.226562500000, 0.056152343750, 0.249267578125, \
                 0.300781250000, 0.268066406250, 0.392089843750, \
                 0.455810546875, 0.309326171875, 0.403320312500, \
                 0.337646484375, 0.632568359375, 0.533203125000, \
                 0.475341796875, 0.331542968750, 0.102783203125, \
                 0.195556640625, 0.262451171875, 0.436035156250, \
                 0.256103515625, 0.227294921875, 0.206542968750, \
                 0.076171875000, 0.112304687500, 0.265869140625, \
                 0.191894531250, 0.256835937500, 0.527832031250, \
                 0.408935546875, 0.451660156250, 0.386718750000, \
```

```
                    0.495849609375, 0.439697265625, 0.505468750000, \
                    0.204101562500, 0.206298828125, 0.239257812500, \
                    0.404541015625, 0.412353515625, 0.431884765625, \
                    0.420410156250, 0.230224609375, 0.176269531250, \
                    0.129296875000, 0.402832031250, 0.267822265625, \
                    0.227539062500, 0.477539062500, 0.505126953125, \
                    0.403808593750, 0.452636718750, 0.556152343750, \
                    0.423095703125, 0.395751953125, 0.310351562500, \
                    0.167968750000, 0.398437500000, 0.471435546875, \
                    0.395019531250, 0.485107421875, 0.437011718750, \
                    0.159423828125, 0.204833984375, 0.247558593750, \
                    0.097851562500, 0.273925781250, 0.338623046875, \
                    0.200683593750, 0.216796875000, 0.376220703125, \
                    0.639648437500, 0.491210937500, 0.340820312500, \
                    0.421386718750, 0.525390625000, 0.482421875000, \
                    0.495605468750, 0.444580078125, 0.336914062500, \
                    0.258789062500, 0.074462890625, 0.229980468750, \
                    0.207763671875, 0.376464843750, 0.352539062500, \
                    0.364062500000, 0.333984375000, 0.476806640625, \
                    0.168945312500, 0.177734375000, 0.209472656250, \
                    0.151855468750, 0.017822265625, 0.527587890625, \
                    0.469726562500, 0.426025390625, 0.436718750000};

// Parameter Vt is different from the one in table 8.1 on page 93 in Cecile
// Cornou's Ph.D. Thesis "Automated Monitoring Methods For Group Housed Sows".
// This Vt originates from distributed data material.

float Vt = 0.007187178;

// Parameter Wt is different from the one in table 8.1 on page 93 in Cecile
// Cornou's Ph.D. Thesis "Automated Monitoring Methods For Group Housed Sows".
// This Wt originates from distributed data material.

float Wt[] = {0.01828878, 0.0000000000, 0.00000000000, \
              0.00000000, 0.0001154555, 0.00000000000, \
              0.00000000, 0.0000000000, 0.00011545550};

void matrix3x3addition(float *a, float *b, float *c)
{ int i;
  for(i = 0; i<9; i++)
  { c[i] = a[i] + b[i];
  }
  additions = additions + 9;
}

void matrix1x3addition(float *a, float *b, float *c)
{ int i;
  for(i = 0; i<3; i++)
  { c[i] = a[i] + b[i];
  }
}

void matrix1x3constantAddition(float *a, float *b, float *c)
{ int i;
  for(i = 0; i<3; i++)
  { c[i] = a[i] + b[0];
  }
  additions = additions + 9;
}

void matrix1x3mul3x1(float *a, float *b, float *c)
{ int i,j;
  for(i = 0; i<3; i++)
  { for(j = 0; j<3; j++)
    { c[3*i+j] = a[i] * b[j];
    }
  }
  multiplications = multiplications + 9;
}
```

```
void matrix3x1transpose3x3multiplication(float *a, float *b, float *c)
{ c[0] = a[0]*b[0] + a[1]*b[3] + a[2]*b[6];
  c[1] = a[0]*b[1] + a[1]*b[4] + a[2]*b[7];
  c[2] = a[0]*b[2] + a[1]*b[5] + a[2]*b[8];
  multiplications = multiplications + 9;
  additions = additions + 6;
}

void matrix3x3multiplication3x1(float *a, float *b, float *c)
{ c[0] = a[0]*b[0] + a[1]*b[1] + a[2]*b[2];
  c[1] = a[3]*b[0] + a[4]*b[1] + a[5]*b[2];
  c[2] = a[6]*b[0] + a[7]*b[1] + a[8]*b[2];
  multiplications = multiplications + 9;
  additions = additions + 6;
}

void matrix3x1multiplication(float *a, float *b, float *c)
{ c[0] = a[0]*b[0] + a[1]*b[1] + a[2]*b[2];
  multiplications = multiplications + 3;
  additions = additions + 2;
}

void print3x3(float *k)
{ //  printf("Matrix:\n");
  printf("   %e %e %e\n",k[0], k[1], k[2]);
  printf("   %e %e %e\n",k[3], k[4], k[5]);
  printf("   %e %e %e\n\n",k[6], k[7], k[8]);
}

void print1x3(float *k)
{ //printf("Matrix:\n");
  printf("   %e %e %e\n",k[0], k[1], k[2]);
}

void printScalar(float *k)
{ //printf("Scalar:\n");
  printf("   %e\n\n",k[0]);
}

int main(void)
{ int i,j;
  float temp1[9], temp2[9], temp3[0];

  // Initialization of Ct
  Ct[0] = 0.147833;
  Ct[1] = 0.000000;
  Ct[2] = 0.000000;
  Ct[3] = 0.000000;
  Ct[4] = 0.147833;
  Ct[5] = 0.000000;
  Ct[6] = 0.000000;
  Ct[7] = 0.000000;
  Ct[8] = 0.147833;

  // Initialization of mt
  mt[0] = 0.3340450;
  mt[1] = 0.0984615;
  mt[2] = 0.3192043;

  // R code converted to C code:
  // ft[1] <- mt[1,1,1];

  ft[0] = mt[0];

  // R code converted to C code:
  // Qt[1] <- t(Ft[1,]) %*% (Ct[,,1] + Wt) %*% Ft[1,] + Vt;

  matrix3x3addition(&Ct[0], &Wt[0], &temp1[0]);
  matrix3x1transpose3x3multiplication(&Ft[0], &temp1[0], &temp2[0]);
  matrix3x1multiplication(&temp2[0], &Ft[0], &temp1[0]);
  Qt[0] = temp1[0] + Vt;
```

```
// R code converted to C code et[1] <- Yt[1] - ft[1];() :

et[0] = Yt[0] - ft[0];

for(i=1;i<120;i++)
{ //at[,,i] <- mt[,,i-1];

  at[0] = mt[0];
  at[1] = mt[1];
  at[2] = mt[2];

  //Rt[,,i] <- Ct[,,i-1] + Wt;

  matrix3x3addition(&Ct[0], &Wt[0], &Rt[0]);

  //ft[i] <- t(Ft[i,]) %*% at[,,i];

  matrix3x1multiplication(&Ft[3*(i%21)], &at[0], &ft[0]);

  //Qt[i] <- t(Ft[i,]) %*% Rt[,,i] %*% Ft[i,] + Vt;

  matrix3x1transpose3x3multiplication(&Ft[3*(i%21)], &Rt[0], &temp1[0]);
  matrix3x1multiplication(&temp1[0], &Ft[3*(i%21)], &temp2[0]);
  Qt[0] = temp2[0] + Vt;
  additions = additions + 1;

  //At[,,i] <- as.vector(Rt[,,i] %*% Ft[i,]) / Qt[i];

  matrix3x3multiplication3x1(&Rt[0], &Ft[3*(i%21)], &temp2[0]);

  At[0] = temp2[0] / Qt[0];
  At[1] = temp2[1] / Qt[0];
  At[2] = temp2[2] / Qt[0];

  divisions = divisions + 3;

  //et[i] <- Yt[i] - ft[i];

  et[0] = Yt[i] - ft[0];

  additions = additions + 1;

  //mt[,,i] <- at[,,i] + At[,,i]*et[i];

  mt[0] = at[0] + At[0]*et[0];
  mt[1] = at[1] + At[1]*et[0];
  mt[2] = at[2] + At[2]*et[0];

  multiplications = multiplications + 3;
  additions = additions + 3;

  //Ct[,,i]<- Rt[,,i] - At[,,i]  %*% t(At[,,i]) * Qt[i];

  matrix1x3mul3x1(&At[0], &At[0], &temp1[0]);

  for(j=0;j<9;j++)
  { Ct[j] = Rt[j]-temp1[j]*Qt[0];
  }
  multiplications = multiplications + 9;
  additions = additions + 9;

  // printf("\nCt:\n");
  // print3x3(&Ct[9*i]);
}
```

```
  printf("Output: \n\n");

  printf("\nat:\n");
  print1x3(&at[0]);

  printf("\nRt:\n");
  print3x3(&Rt[0]);

  printf("\nft:\n");
  printScalar(&ft[0]);

  printf("\nQt:\n");
  printScalar(&Qt[0]);

  printf("\nAt:\n");
  print1x3(&At[0]);

  printf("\net:\n");
  printScalar(&et[0]);

  printf("\nmt:\n");
  print1x3(&mt[0]);

  printf("\nCt:\n");
  print3x3(&Ct[0]);

  printf("Additions: %i\n", additions);
  printf("Multiplications: %i\n", multiplications);
  printf("Divisions: %i\n", divisions);
  return 0;
}
```

Chapter 12

Conclusions

This thesis concerns the system-level modeling and design of networked multiprocessor embedded systems. The contributions of this thesis include the following concepts and techniques:

The first part of the thesis presents an overview of the existing theories and practices of modeling and simulation of multiprocessor systems-on-chip. The systematic categorization of the plethora of existing programming models at various levels of abstraction is the main contribution here which is the first such attempt in the published literature.

The second part of the thesis deals with the issues related to the development of system-level design methodologies for networked multiprocessor systems-on-chip at various levels of design abstraction with special focus on the modeling and design of wireless integrated sensor networks which are an emerging class of networked embedded computer systems.

The work described here demonstrates how to model multiprocessor systems-on-chip at the system level by abstracting away most of the lower-level details albeit retaining the parameters most relevant at the system-level. The multiprocessor modeling framework is then extended to include models of networked multiprocessor systems-on-chip which is then employed to model wireless sensor networks both at the sensor node level as well as the wireless network level.

In the third and final part, the thesis covers the issues related to the design, implementation and testing of a system-on-chip wireless sensor node development platform, specifically, for the Hogthrob project. This part also deals with the cycle-accurate model of the multiprocessor system-on-chip and its possible extensions to the transaction-level MPSoC model.

An important design consideration while designing our sensor node development platform, called the Hogthrob platform, was to reduce the overall cost of prototyping by using COTS (Common Off-the-Shelf) components. Another consideration was to have the capability to experiment with various combinations of sensors, radio transceivers and microprocessors to select the optimal combination. To achieve this objective, we needed to adapt a modular design strategy so that we could swap sensors and radio transceivers with the ones resulting in more efficient energy and system performance. For trying different microprocessors and/or to perform hardware acceleration, we needed some form of reconfigurable logic on the wireless sensor node so that we could configure it with various mircoprocessor cores. Of couse, low power, small form factor, and robust packaging were necessary as well because the wireless sensor nodes have to be mounted on sows.

The Hogthrob platform has also been designed with a view to explore the tradeoffs of implementing application functionality either in software (on the embedded processor) or hardware (on the reconfigurable logic/custom processor), without being constrained by the initial design choices. The hardware and software components that constitute a sensor network system had to be optimized so that they meet the resource and energy constraints while delivering acceptable performance. To meet these objectives, we needed to adopt a hardware/software codesign perspective for designing the wireless sensor nodes which could be customized to suit our application. As an initial design step, all the application functionality was placed on the embedded processor and was gradually moved to the FPGA. At the initial stage of software development, the radio transceiver and other peripherals were being controlled by the software running on the embedded processor but, eventually, the embedded processor only served to initialize the FPGA and function as an external timer and an A/D converter for the FPGA.

Because application-specific integrated circuits (ASIC's) can clock at much lower speeds and use less numerical precision, they consume several orders of magnitude less energy than the programmable processors. While the line between dedicated processors and general-purpose (more easily programmed) processors is constantly shifting, generally speaking, a mixed architecture was needed for computational subsystems dealing with connections to the physical world. The ratio in die area between the two approaches - ASIC and programmable processor - scales with the technological changes, so ASIC's maintain a cost advantage

over many chip generations. Convenient programmability across several orders of magnitude of energy consumption and data processing requirements is a worthy codesign research goal for pervasive computing. In the meantime, while the codesign researchers continue to pursue that goal, multiprocessor systems are needed in the wireless integrated sensor networks.

The thesis, as a whole makes contributions to the field of research by describing a design methodology for networked multiprocessor embedded systems at three layers of abstraction from system-level through transaction-level to the cycle accurate level as well as demonstrating it practically by implementing a wireless sensor node design.